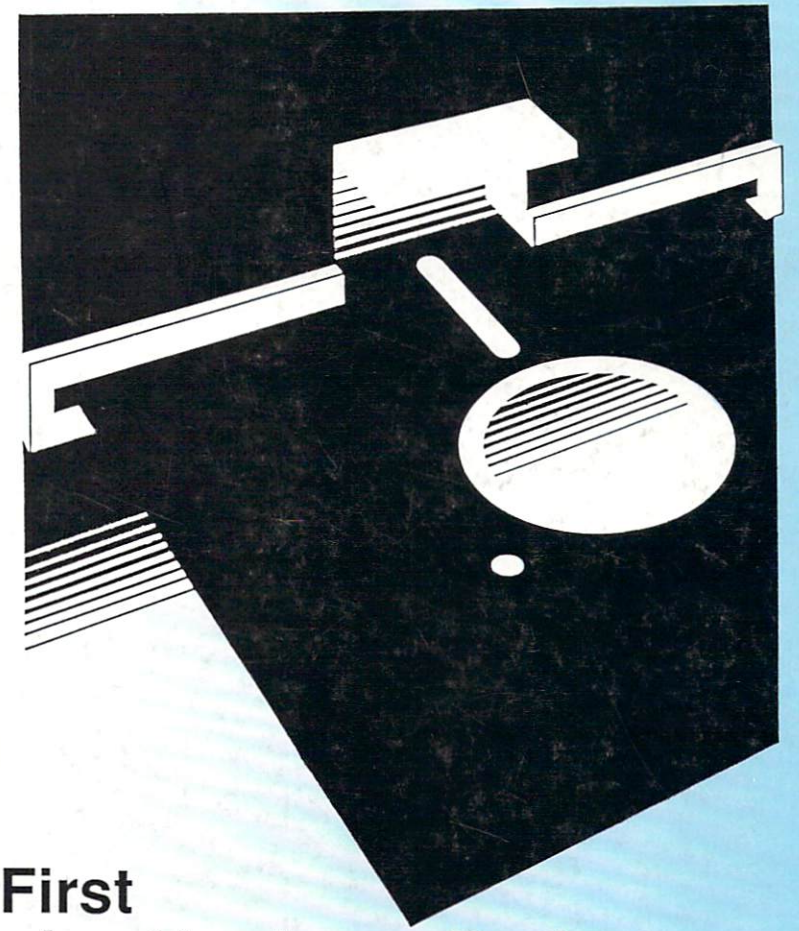
Amazing Computing™

May 1986
Vol.1/ Number 4
U.S.A. $2.50
Canada $3.50

Commodore Amiga™ Information and Programs

SCRIMPER!
Screen Image Printer in C!



Roomers
Miga Mania

Reviews
Skyfox
Arcticfox

Tutorials
Forth
C, Basic

AMICUS
CD ROM

Our First
Amazing Hardware Project:
Connect your own IBM 5 1/4 Drive

Amazing Computing

&

Amazing Dealers

The following are **Amazing Dealers**, dedicated to supporting as well as selling the Commodore-Amiga™. They carry **Amazing Computing**™, your resource for information on the Amiga™.

If you are not an **Amazing Dealer**, but would like to become one, contact:

PiM Publications
P.O. Box 869
Fall River, MA. 02722
1-617-679-3109

Alabama

Abax Data Systems	Huntsville
Command Computers	Birmingham
Madison Books & Computers	Madison
Mel's Photo & Computer Shop	Montgomery
Universal Computer System	Mobile

Arizona

Computer West	Pheonix
Copperstate Business Systems	Pheonix
North American Digital	Tucson
Tronixtools	Tucson

Arkansas

SIS Inc.	Mt. Home
The Micro Shop	Little Rock

California

Brown Knows Computing	Redlands
C.F.T.	Clairemont
Century Computer Systems	La Habra
Chambers Computer Supply	Studio City
Computer Attic	Palo Alto
Computer Junction	Sanata Ana
Computer Nook, Inc.	San Bernadino
ComputerLand Freemont	Freemont
ComputerLand of Citrius Hts.	Citrius Hts.
ComputerLand of Sacramento	Downtown
ComputerLand of Sacramento	Howe Ave.
ComputerLand of Stockton	Stockton
Computer Literacy Book Shop	Sunnyvale
Home Computing Center	San Bruno
K J Computers	Granada Hills
Ridgecrest Computer Ctr. Inc.	Ridgecrest
S.O.S. Computers	Los Angeles
Software First	Santa Rosa
The Computer Room	Scotts Valley
The Floppy Disk Inc.	Downey
Winner's Circle	Berkley
HT Electronics	Sunnyvale

Colorado

Citadel Computers	Colorado Springs
Computer Room	Aurora
Ideal Computer Systems	Evergreen
Micro World	Lakewood
Sun Country Computers	Colorado Springs
Whole Life Distributors	Englewood

Connecticut

Connecting Point	Georgetown
Mnemonics Inc.	Norwalk
Personal Computer Center	Norwich
Software Kingdom P.O. A1007	East Windsor
Spectrum Computers	New Haven

Florida

Commodore Computer Shop	Ocala
Computer Bar	Pennsacola
Computer Image	Miami
Computer Terminal Inc.	Sebring
Computer Ware Inc.	Tamarc
Computers 4 Rent	W. Palm Beach
Computers Plus	South Daytona
Cotronics Inc.	Stewart
Education Computers Etc.	Talahasse
Family Computers	Sarasota
Florida Book Store	Gainsville
Gulf Coast Computer Elect.	Panama City
Megaport Computer Center	Venice
Micro's Etc.	Altamonte Springs
MSI Business System	Brooksville
New Age Electronics	St. Petersburg
Orlando Electronics Co.	Orlando
Rainbow Computr Center	Orlando
Random Access Computers Inc.	Ft. Walton Beach
Softwalls Computer Center Inc.	Holly Hill
The Open Door	Cocoa
The PC Collection	Coral Gables

Georgia

Future System	Augusta
Software South Inc.	Savannah

Iowa

Easy Keys Computer Center	Iowa City
Micro Computer Applications LTD	Marshalltown

Idaho	
ABI Computer & Video	Nampa
Illinois	
BA Business Computers	Batavia
Computer Resolution	Champaign
Computerland O'Hare	Park Ridge
ComputerLand of Niles	Niles
Illini Microcomputers Inc.	Naperville
The Memory Expansion	Normal
Unique Computer	Sterling
Indiana	
Burkat Computer Ctr.	South Bend
Bytrex Computer	Ft. Wayne
Computer Corner	Fort Wayne
Computer People Inc.	Michigan City
Greg Chaney	Greenfield
Micro Computers Inc.	Indianapolis
Vons Computers	West Lafayette
Kansas	
Data Bank Corp.	Wichita
J & L Electronics	Liberal
MidKansas Computers	Newton
Midwest Computers	Manhattan
Thoroughbred Computers	Topeka
Kentucky	
MicroAge Computer Store	Lexington
Reality World Computers	Lexington
Louisiana	
Computer Time	Shreveport
Modern Business Machines	Metairie
Software Center International	Metairie
Software Mart	Metairie
The Computer Clinic	Lafayette
Massachusetts	
Club Computer	Cambridge
Computerland	Leominster
HCS Computer Center	Marshfield
LCA Video & Computer Ctr.	Norwood
Memory Location	Wellsley
OmniTek Computers	Tewksbury
Pioneer Valley Data Equipment	Amherst
Tech Computer Store	Cambridge
The Bit Bucket	West Newton
Tycom Inc.	Pittsfield
Maryland	
Computer Crafters	Wheaton
ComputerWorld	Ocean City
Micro Computer Center	Baltimore
Software Advantage	Rockville
The Logical Choice Inc.	Baltimore
Maine	
Computer Barn	Damariscotta
Valley Computers	Auburn
Michigan	
Basic Computer Center	Grand Rapids
Bits-Bytes-Nibbles	Petowskey
Chelsea Computer	Ann Arbor
Computer Supply Co.	Traverse City
Direct Access	Lansing
Edmunton Computer Center	Pontiac
Galaxy Computers	Allen Park
Michigan Software	Novi
Midland Computer Shop	Midland
Pro-Video	Ukenos
Proffessional Computer Systems	St. Joseph
Programs Unlimited	Grand Rapids
Roseville Computer Store	Roseville
Slip Disk	Madison Heights
Software Plus	Grand Rapids
Specialists Computer	Livonia
The Software House	Kalamazoo
Ye Olde Computer Shoppe	Ypscilanpi
Retail Computer Center Inc.	Farmington Hills
Minnesota	
Computer Place	Minneapolis
Computer Specialties Plus	Monticello
Computers, etc.	Eagan
MCD of Hibbing Inc.	Hibbing
OnLine Computers	Minnetonka
Specialists IM	Hopkins
Missouri	
Associated Computer Services	Springfield
Brands Mart Computers	Kansas City
Commputer Concepts	Columbia
Data Byte	St. Louis
Systems Plus	Independence
Mississipp	
Enterprises Unlimited	Jackson

Montana	
Appelgren Computer Services	Great Falls
North Carolina	
Digitz	Raleigh
TDS Computers	Carrboro
North Dakota	
Computer Associates	Fargo
The Computer Store	Minot
Nebraska	
BULLS-EYE	Lincoln
Double E Electronics	Omaha
New Hampshire	
Computer Mart of New Hampshire	Nashua
New Jersey	
Family Computer Centres	South Orange
Family Computer Centres	Fairfield
Family Computer Centres	Ridgewood
New Mexico	
Academy Computers	Albuquerque
New Horizon Computer System	Alamogordo
Page One Newstand	Albuquerque
Technical Concepts	Les Crues
Nevada	
Century 23 Inc.	Las Vegas
Computer World	Las Vegas
ComputerLand of Carson	Carson City
New York	
Amuse	New York
Byte Shop	Merrick
Castle Computer	Latham
CIA Software Center	Flushing
Compurite Center	White Plains
Computers Etc.	Syracuse
Computer Outlet	Jamestown
Leigh's Computers	New York
Ray Supply Inc.	Glen Falls
Software & Such	Scotia
Software Supermarket	Kenmore
Star Tech Systems	Massena
Tedrow Business Products	Rochester
The Working Computer	Stoney Brook
Video Computer Center	Rome
World Computers Inc.	Hicksville
Ohio	
Computer Network	Newark
Computers Plus of Ohio	Upper Sandusky
Earthrise Micro Systems	Delaware
Earthrise Micro Systems	Columbus
Earthrise Micro Systems	Columbus
Lakes Consumer Electronics	Akron
Micro Center	Columbus
Microwave Magic	Fairfield
North Coast Programming	Willoughby
Quality Computer Appliances	Toledo
Quality Computer Appliances	Toledo
Saxtan Computer Center	Sandusky
Softwaire Centre International	North Olmsted
Software and More	Cincinnati
Oklahoma	
Colonial Video & Computer	Bartlesville
P C Tech	Tulsa
Second Hand Software	Oklahoma City
Video-Comp Inc.	Lawton
Oregon	
Coackamas Computers	Coackamas
Computron Business Systems	Portland
Computron Business Systems	Tigard
IB Computers	Portland
Software Center	Beaverton
Software Express	Eugene
Pennsylvania	
Alpha Omega EBE	Warren
Basic Computer Systems	Hermitage
Data Softique Computerware	Pittsburg
Pittsburgh Computer Store	Pittsburgh
Triangle Computers	Indiana
Rhode Island	
CompUtopia	Providence
CompUtopia	Warwick
Micro Limits	Smithfield
Micro Limits	Warwick
International Computer Services	Johnston
South Carolina	
Avcom International Inc.	West Columbia
Horizon	Greenville
Horizon	Greenville

Tennessee	
ComputerLab of Memphis	Memphis
Eastern Computer Inc.	Knoxville
Software First	Nashville
Texas	
Byte by Byte	Austin
C.E.M. Corporation	Brazoria
Colonial Video	Houston
Colonial Video	Houston
Computer Age	Dallas
Computer Age	Houston
Computer Concepts	Waco
Computer Magic	Austin
Computer Revelations	Houston
Computer Time Inc.	Marshall
Metroploitan Computer	Richardson
Micro Search	Huston
Micro Search Inc.	Houston
Regency Educational Systems	Dallas
Softex	Watauga
Softwaire Centre	Dallas
Software Terminal	Fort Worth
The Computer Experience	San Antonio
Thye Computer Shop	Abaleen
Utah	
Computers Plus	Murray
Virginia	
Colony Ltd.	Hampton
Computer Annex	Morgantown
Diskovery	Falls Church
Family Computer Center	Fairfax
Virginia Micro Systems	Woodbridge
Washington	
A.P.P.L.E. CO-OP	Renton
Bob Mascott	Seattle
Com-Soft Computers	Everett
Computers +	Tacoma
Computron Business Systems	Yakima
Micro Age Computer Store	Bellview
Programs Plus	Seattle
Wisconsin	
Colortron Computers	Racine
Software Shoppe	Fond Du Lac
TMW Software Inc.	Wavsan
Canada	
Compute or Play	Alberta
Computerworks	Alberta
Computer Shop	Alberta
Data Pro	Alberta
Computer Store of Calgary	Alberta
The Software House	Alberta
Hindsin Computing	Alberta
Red Deer Micro Systems	Alberta
Memory Lane	Alberta
Computer King	British Columbia
Conti	British Columbia
Nutec	British Columbia
Sprite	British Columbia
Strider	British Columbia
Super Software	British Columbia
Canadian Computer Era	Manitoba
Micromart	Manitoba
ComputerWorld	Maritimes
J. W. Allen	Maritimes
Kobetek	Maritimes
Minicomp	Maritimes
Arkon	Ontario
Chatam Computer	Ontario
ComputerLand	Ontario
Comspec	Ontario
Electronics 2001	Ontario
Hastings Data	Ontario
The Computer & You	Ontario
Thornhill	Ontario
Software Co.	Quebec
Deltronic Solare	Quebec
Mierodata	Quebec
Info Mereler	Quebec
Laval Micro	Quebec
Vision Tronique	Quebec
Distribution Visipro	Quebec
Distributed in Canada by	
Avita Software Distributors & Phase 4 Distributors Inc. 1-800-661-8358	Concord, Ontario Southeast Calgary, Alberta

PUBLIC DOMAIN SOFTWARE

PiM Publications, Inc. has
AMICUS Disks 1 through 8
and
Fred Fish Disks 1 through 11
available at special prices

Per disk
$6.00 to Amazing Computing Subscribers
or
$7.00 to non subscribers
(MA Residents add 5% sales tax)

Make checks payable to:
PiM Publications Inc.
P.O. Box 869
Fall River, MA 02722

Everyone is encouraged to distribute this software freely to your friends, members, and customers.

Please allow 4 weeks for delivery

Amazing Computing

Publisher: Joyce A. Hicks
Circulation Manager: Doris Gamble
Assistant to The Publisher: Robert James Hicks
Corporate Advisor: Robert H. Gamble

Managing Editor: Don Hicks
Hardware Editor: Ernest P. Viveiros
AMICUS Editor: John Foust

Amazing Authors:
John Foust
Don Hicks
Kelly Kauffman
Perry Kivolowitz
Richard Miner
George Musser Jr.
Daniel Zigmond
&
The Amigo

Special Thanks to:
Robert H. Bergwall
RESCO, Inc.
David Street
E.P.V. Consulting
New England Technical Services
Interactive Tutorials Inc.

Advertising Sales:
1-617-679-3109

Amazing Computing™ (ISSN 0886-9480) is published by PiM PUBLICATIONS, Inc., P.O. Box 869, Fall River, MA. 02722. Subscriptions: in the U.S., 12 issues for $24.00; Canada and Mexico, $30.00; Overseas, $35.00. Printed in the U.S.A.

Amazing Computing

Table of Contents

May 1986

From the Editor:

by Don Hicks

May
Welcome to the fourth issue of Amazing Computing. This is a super issue for us. We have packed the magazine with beginner and advanced user information.

April Fool
I hope all of our readers appreciated the light touch we placed on the April issue's front cover. We thought a few misspelled words might bring smiles. Listen, this is supposed to be fun and no one in their right mind would misspell a word on the front cover. Would they?

New Authors
We are beginning to see the effects of our search for "new" writers. We have been asking for contributions of articles and tips to the magazine, and they are now coming in quickly.

However, there is always room for **your** contribution.

We are not asking for only the experienced and previously published author, **we want you**. We want your enthusiasm and your growth. Your ideas and practice. Remember, anything you believed was important to place on the Amiga, is important to our other readers. Our pay rates will not make you rich, but they are a nice reward for doing what you already enjoy.

Amazing Computing is excited about the prospects of you publishing your deeds. Let us see them.

CD ROM
Jim O'Keane has entered his first article for AC. Jim has written a report on the recent CD Rom conference by Microsoft. In a hope to better understand this new technology , Jim has given us an insider's perspective of the technology and the conference.

Our First Hardware Project!
Our hardware editor, Ernest Viveiros, has written his first hardware article for Amazing Computing. Working with information supplied by a reader in Maine, Jim Rutherford, and our AMICUS editor, John Foust, E.P.V. was able to construct an interface between a Shugart IBM 5 1/4 drive and the AMIGA. It utilizes the Transformer software from Commodore. And Works!

This is Mr. Viveiros' first article, and although short on words, he has conveyed exactly what he wanted to say with schematics and graphics.

Basic
Both Rick Wirch and Kelly Kauffman have supplied us with insight into the Amiga's Basic. Their research and articles should help a few of us along the way to better understanding our machines.

C
John Foust has taken a look at the new compiler from Manx, Azteck C. His part review and part tutorial should be of interest to the C users in our group.

Scrimper!
Not to be out done, Perry Kivolowitz has contibuted a great new program to the C community, scrimper!. In an article that is part listing, part tutorial, Perry has brought a little reason to the tangle mess some of us feel when manipulating the hidden controls of our Amigas. Although the listing is long, we feel information should be of special interest.

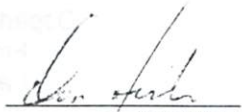
Entertainment!
Mr. Bobo has returned with a close look at two releases from Electronic Arts, Skyfox and Arcticfox. There are some pure breds who refuse to admit their Amigas are ever used for games. Well, if that is true, great. But remember, some of the better advancements in graphics and speed have been through the use of play. To be able to play and exercise both the machine's capabilities and ours, allows us to extend both.

I am reminded of the Apple II. It was supposed to be an excellent game machine. It was through the use of its graphics and capabilities that an entire new way of working with computers was discovered. Doesn't it seem obvious we should use our play to further ourselves. And besides, after careful study and work, I finally won at Arcticfox and these are the same reasons I used to explain my time spent before the game to my boss, Joyce.

Comdex
As I write these words, we are preparing a trip to Atlanta Georgia and Comdex. For the unitiated, Comdex is a meeting of Suppliers and Dealers in the Computer market. It is the stage where each supplier and manufacturer must try to outshow the other. The strength of the competition is in showing the superiority of their product in one way or another over everyone else.

Many Amiga developers have told us to "see us at Commodore's booth at Comdex." We have no idea what to expect, since this will be our first Comdex, but I am certain it is worth note and will be the central piece of our June issue (besides, I have never been to Atlanta).

See you then, and keep those articles coming.



Don Hicks
Managing Editor

This assembler code can benefit programmers who want to see the actual machine language the compiler is producing, to search for bugs, or to learn more about the way a compiler works.

In comparison, the Lattice compiler produces an intermediate file, with a filename extension of '.q', called a "quad file," which becomes an object file after the second pass of the compiler.

There is a Lattice utility called 'omd', for Object Module Disassembler, which can disassemble a Lattice object file into assembly language source.

I noticed one simple difference right away. I had made a series of simple text filters using the Lattice compiler, and I compiled them with the Manx compiler. I was amazed at the code size reduction. Some of these filters were 18 K with Lattice, but only 6 K with Aztec.

However, I noticed the last few lines were missing from each text file I passed through the filter. I needed an explicit 'fclose()' in the filter to properly close the output file, in the Manx version.

Apparently, Lattice's implicit exit() at the end of a program did this for me, while Manx did not.

('Filter' is a common name for a program that converts one file to another, translating it in some way. This filter removes carriage-returns and other spurious characters.)

Small programs

The code in Listing One compiled in a 3:05 minutes with the Lattice compiler, and the executable load file was 10492 bytes. Under similar compilation conditions, the Manx program compiled in 1:00 minute, and the program was 3716 bytes.

Please note that these times reflect having the Manx library in RAM:, while the Lattice libraries were not. Both compiles were driven by a simple EXECUTE batch file. More on this later.

By linking the Lattice amiga.lib before lc.lib, the code size went from 14944 to 10492 bytes, since amiga.lib contains some functions nearly identical to functions in lc.lib.

I remembered that Lattice adds symbol names to the load file, by default, while Manx has an option to do so. The StripA program, supplied to developers on the Wack disk, removes the symbols from a load file. Using this, the Lattice file above was reduced from 10492 bytes to 9932 bytes.

I knew this comparison was a little harsh on Lattice. I decided to rewrite the filter, and try again.

Instead of standard C calls, I used AmigaDOS calls. This should be a better measure, since both compilers simply make calls to the AmigaDOS routines, and the overhead would stand out more than code that was fattened by standard I/O routines. See Listing Two.

I had to use the '-v' option on 'lc2', to turn off runtime stack checking, to remove two unresolved externals in the link, __xcovf and __main. I linked Astartup.o instead of Lstartup.o.

Surprisingly, the Lattice version of this, after StripA, was 1516 bytes. The Manx version, with the same code, was 1760 bytes. The compile times were similar to the results given in the standard I/O example.

RAM: disk use

The compiler uses the RAM: disk effectively, depending on environment variables. The default startup disk is usable right out of the package, and a simple SET statement will shave quite a bit of time from each link.

Manx has an interesting feature that can shave off another thirty seconds or so off most compile times. Most often, the '#include' files for a program do not change over time, it's the program source code that changes most often.

For every compile, the compiler has to scan and parse the same old '.h' files. The Manx compiler can 'precompile' the include files for a program, and write the internal compiler symbol table to disk.

Later on, on subsequent compiles, the compiler can read in

Listing A

	Manx		Lattice	
	Manx libs on disk	Manx libs in RAM:	libs in RAM:, precompiled .h files	libs on disk, Lattice with Alink libs on disk
FASTER switch				
'cc' or 'lc1' starts	0:15 (1:15)	0:15 (1:15)	0:15 (0:30)	0:20 (2:40)	0:20 (2:40)
'as' or 'lc2' starts	1:30 (1:00)	1:30 (1:05)	0:45 (1:10)	3:00 (0:45)	3:00 (0:45)
'ln' or Alink starts	2:30 (3:20)	2:35 (1:00)	1:55 (1:10)	3:45 (2:45)	3:45 (1:45)
Compile ends	5:50	3:35	3:05	6:30	5:30

Resulting load file size

Manx: 13,564

Lattice: 26,440 (27,496 before StripA)

the pre-parsed symbol table, and load it into the internal tables, and begin the compilation process with a jump on the game.

In most Amiga programs that interface to the Intuition and the operating system, the sheer size of the include files is often greater than the source code itself. This feature can save a large proportion of time, compared to the programming effort required.

Large programs

To compare compile speeds for larger programs, I used the Amigaterm telecommunications program by Michael Mounier. This is roughly 32 K of source. It is one of the dozen or so example programs provided on disk two of the Manx system. The results are posted in Listing A above.

What does this all mean? On the surface, Manx appears to be far superior, at least in terms of compile time and code size. Lattice looks slow and fat.

Looking deeper, Lattice clearly suffers from large, fat standard C routines. In the pure AmigaDOS program, Manx and Lattice might come out equal, as in the filter example above. But Manx has smaller library routines, and that can make a difference.

You can remove another few K from a Lattice program by turning off stack checking. It seems as though there are a dozen little optimizations one can perform on a Lattice program, just to make it the same size as Manx. Why should a programmer sacrifice standard C routines for AmigaDOS routines?

I have not mentioned executable program speed. I have heard many reports that Aztec programs are much faster than equivalent Lattice programs, but I cannot verify these claims. Conventional wisdom suggests that a smaller program will run faster, given the same C code, and I'd agree.

Speed comparisons are a large, rusty can of worms that I'll leave to another writer, in another column. I worry that speed comparisons often deteriorate into that field of advertising called "benchmarketing." Just like the Intel vs. Motorola debates...

Manx assembler

The Manx assembler is invoked during every compile, so the times in the listings should demonstrate that it is fast.

It is a macro assembler, but its largest shortcoming is incompatibility with the Metacomco assembler. This isn't Manx's fault, of course, but only a detriment to Amiga developers. The Manx assembler does not have as many assembler directives as the Metacomco assembler, so converting the assembler header files from Lattice to Manx can take quite a while.

Of course, another shortcoming of Manx in general is that the object files are incompatible with the Metacomco linker. Considering the snail's pace of Alink, a.k.a. "Deep Thought", this might not be so bad. I have heard a rumor that Manx will have a conversion utility for this in the future.

Disk organization

The Manx disk is usable right from the shipping box. This saves the time and headache to organize the files into a manner conducive to quick compiles.

The Lattice compiler requires a tedious re-organization of the original disks. This includes optimizing the EXECUTE batch files for use of the RAM: disk, for example.

With the first version of Lattice, many developers decreased compilation times by stripping the comments from the '#include' files. This increased the amount of free space on the disks, and eased the job of the compiler.

This sort of out-of-the-box organization is present in the Lattice compiler for the IBM PC. It has an install batch file which creates the proper directories on your hard disk, and copies the files from the distribution disks. The organization of the Manx disks is appreciated.

Portable code
While C is often praised as a portable language, people who program in C know this is often far from the truth. It may be possible to write portable code, but many programmers are lazy, and assume that a program will never be ported. The simple rules of portable code are often violated for the sake of quickly generated code.

There are several incompatibilities of this sort between Manx and Lattice C. The most major difference is the default size of an 'int'.

Manx thinks that an 'int' is sixteen bits, while Lattice thinks an 'int' is four bytes, or 32 bits. Manx and Lattice agree that a 'long' is 32 bits. AmigaDOS functions use a 32 bit data type.

What Lattice considers to be a 'short', Manx considers an 'int'. A Lattice programmer who assumes 'int' is a generic storage type for every data type will have troubles porting to Manx.

For example, a Lattice programmer knows that a Lattice 'int' will hold a 68000 address, which requires 32 bits, and might use this assumption in a program implicitly. This use only becomes explicit under close examination of the code.

Remember, C will let you assign almost anything to anything, so a programmer might try to store an address value in an 'int' variable, which doesn't work when an 'int' is 16 bits, and an address is 32 bits.

Of course, one could argue that 32 bits is a natural size of data on the 68000, and that 'int' should be the most natural data type on a machine, so they should be the same on the Amiga. This is the opinion of the Lattice manual, and many C programmers.

+I option
Actually, Manx knew this, and included a command-line option, to tell the compiler to make 'int' a 32 bit data type. This will resolve most conflicts between Manx and ill-written Lattice programs.

Incompatibilities can still occur when the size of a 'struct' is to stay the same, and the 'struct' includes 'int' or 'short' types. The latest revision of the C language standard allows passing a struct as a return value from a function. Manx does not allow this, but Lattice does.

Manx does not have enumerated types, while the latest Lattice compiler does. The first Amiga Lattice compiler professed to allow 'enum's, but the compiler would always Guru Mediate when it found one.

Another problem is in the declaration of argv[], the standard method of passing commmand line arguments from the operating system to a C program. As developer D.J. James pointed out to me, Manx only allocates enough elements in this array to hold the number of arguments from the command line, and no more.

Lattice declares a static number of array elements for argv[], and so a not-so-smart programmer might use the extra, unused pointers-to-char elements of the array under Lattice, but the program wouldn't work without modification under Manx.

For the sake of this example, assume 'argc' is 3, which means argv[0], argv[1], and argv[2] hold pointers to character strings. argv[3] is available to hold another pointer under Lattice, but not under Manx.

Compiler-specific functions
Also, compilers are often supplied with a large number of compiler-specific functions, and those will always be incompatible.

For example, all Lattice compilers include routines for many string manipulation functions. Most are easily written from scratch, upon closer examination. The Lattice names are cryptic, but somewhat regular, since the name of the function serves as a mneumonic to the type of its return value.

For example, Lattice has a 'stpchr()' function, that searches a character string for a given character, and returns a pointer to it, if found.

The 'p' in the name says it returns a pointer, the 'st' says it is a string function. The corresponding Manx function is called 'index()' - a little more easy to understand, but it doesn't describe the return value.

Manx has a set of extended functions, too. Most are similar to Unix system calls, and some enhance the Amiga operating system. One set provides a method for starting new processes in a nicer way than the AmigaDOS functions.

Manx has an 'fexecl()' call that loads and calls another program, and returns its exit code. This capability is lacking in the Execute() AmigaDOS call, which doesn't return the error exit code. Also, Execute() will fragment memory, in its present incarnation. It also loses about fifty bytes of memory every time it is called.

Aztec C can use this flexibility in the size of 'int' to an advantage. Sixteen bit operations are smaller and faster on the 68000, and if a program can get along with 16 arithmetic, it can run faster, in a smaller code size. However, if you need 32 bit 'int' arithmetic, you need to link with a different set of system libraries, to include the 32 bit operators.

(If you think it is peculiar for this situation to occur, the key to this is the C language definition. It says that 'long' should be no smaller than 'int', so both compilers are correct in their assignments of 'int' and 'long' to 32 bit types.)

The commercial version of Manx includes a series of Unix-like utilities. The manuals are very clear about the incompatibilities of each tool as compared to the Unix equivalent tool.

This is a nice touch, and makes a Unix-head feel right at home. The documentation is specific enough to list which Unix command line switches are implemented for each program.

Inline assembly code

The Aztec compiler allows in-line assembly language, separated by '#asm' and '#endasm' directives. This is nice for quick and/or dirty coding, but even the Aztec manual recommends that assembly language routines should be placed outside of C source code.

Of course, inline code is highly non-portable. Often, it depends on the register usage assignments of a particular compiler, and these assignments can change.

Register usage

Aztec C uses 68000 registers D0-D3 for register variables, and registers D4-D7 for intermediate values. It uses registers A0 and A1 for temporary values, and A5 is its stack frame pointer. In small data models, A4 points to the middle of the data segment. A6 is for temporary values, with A7 pointing to the top of the stack.

Lattice uses a different set of registers than Manx, and some programmers have pointed out that Manx uses a strange set of registers, as compared to other 68000 C compilers. This only hinders a few applications, where a programmer could use a more efficient, optimizing compiler on some other 68000-based machine, and transfer the code to the Amiga.

Large and small models

Aztec allows small and large code and data models. This doesn't mean the size of a block of code or data is limited, as in the 8086 family of processors. Instead, these are the basis for code and data optimizations.

A small code model means that call or jump instructions must be within 32 K bytes of their destination, since position-independent instructions are used, specifying the destination as an offset from the PC.

An address register will be reserved, and it will point to the middle of the code segment, so instructions can use it as a base address, and supply an offset from that base, to form a destination address.

However, if a small code model function needs to reference a location beyond 32 K bytes away, the linker will build a jump table, which means the function will use the jump table as a stepping stone to the distant location.

A large code model program uses position-dependent references for jumps and calls, these are resolved at load time by the normal executable file loader.

This also means that a large model program will take longer to load, since the operating system loader must patch the absolute (or position-dependent) references, adding the load address (base address) of the code segment to each jump, call or data instruction, which carries an offset.

In its translation of the C source code, the compiler can optimize the machine code to use relative instructions, using the address of the center of the segment as the base address, and supplying a constant offset from that base, plus or minus 32 K bytes. The 68000 has shorter, faster instructions for branching and data manipulation that use this sort of base and offset indexing.

A small data model program is limited to 64 K of contiguous global and static data. A data register will be reserved to point at the middle of this segment, and all data references will be offset from that register.

A large data model program can have an unlimited sized data segment, since absolute instructions will be used to reference the data.

The Manx manual explains these tradeoffs very well. The use of overlays is discussed in similar detail.

Other options for 'cc' include setting the size of many compiler symbol tables, including the expression, symbol abd 'case' tables.

The preprocessor also pre-defines __FILE__, __LINE__, __FUNC__ symbols, very useful in debugging. These expand to the filename, line number, and currently-compiling function name, respectively.

CHIP or FAST memory

The Aztec linker allows the programmer to specify whether code and data for each link module should be placed in CHIP or FAST memory on loading.

The first 512 K of memory you have in your machine is MEMF_CHIP memory. This is the only memory accessible to the video hardware, so all video images in use must be placed in this memory. Additional memory beyond 512 K is called MEMF_FAST memory, or just FAST memory for short.

For example, a program might have several icon images it needs in video memory. But the rest of the program should be in FAST memory, if possible, to leave more video memory for other programs.

The FAST memory is considered fast because it sees no contention from the video chips. The video chips will sometimes "steal cycles" from the processor, in order to manipulate CHIP memory. Programs executing in CHIP memory will run slightly slower at these times.

The Lattice programmer has to use the ATOM utility to specify CHIP or FAST memory, after compilation and linking.

This is an admitted temporary fix for Lattice users. ATOM stands for "Amiga Temporary Object Modifier".

The ATOM utility massages an object file, so this must be done after every compile and link. So far, few developers are constrained by this, but it is a serious theoretical problem, for future Amiga programs that take advantage of several megabytes of memory.

Manual details

The manual appears to be missing a chapter on system-specific dependencies. One section of the manual referenced this chapter, as I noticed when I was searching for information on the Aztec implementation of argv[].

The first few pages of the manual are a tutorial. It covers the basics of formatting a disk, and continues with "Think of a disk as a closet where we want to store different sorts of items". Then it explains 'cd'.

Why bother including this? Fortunately, it doesn't last long, and the rest of the manual does not retain this attitude. The manual is generally clear, and very explicit in the areas of interest to systems-level programmers.

(This reminds me of microprocessor manuals that start with "microprocessors only understand ones and zeroes", and by page 19, its talking about "the sign-extended displacement integer in the low-order eight bits of the extension word.")

(Another prime offender is the AmigaDOS Developer's Manual. Its first few pages include whimsical instructions explaining how to write programs on the Amiga. To paraphrase, it says "Write your program, compile it, link it, and load it, and watch it run!" I'm not joking, and I'm not leaving many words out, either. Were they trying to fill space?)

The Aztec manual has an extra 'style' chapter. It starts out with obligatory handwaving about the virtues of structured programming, and continues by explaining the difference between '=' and '=='.

Yawn, again. Neither experienced or novice programmers need this. A recounting of C programming cherry-pitfalls and style suggestions can, and do, fill several books. Novice programmers should not expect a compiler manual to be a tutorial in C, and most heavy-duty programmers are reticent to change their C style.

Summary

Overall, the Manx Software Systems Aztec C compiler system is aimed directly at the Amiga software market, and it should edge out the Lattice compiler for many developers. Manx code is small and fast, and its compile times are far superior to the Lattice compiler.

(Listing One)

```
/* This filter changes CR/LF to LF, and removes binary.
 *
 * Usage:
 * crlf2lf < infile > outfile
 *
 */

#include "stdio.h"

main()
{
int c;

  while ((c=getchar())!=EOF) {
    if (c>31 && c<127)
      putchar(c);
    else
      if (c==10 || c==9)
        putchar(c);
  }
  fclose(stdout);
}
```

(Listing Two)

```
/* This filter changes CR/LF to LF, and removes binary.
 *
 * Usage:
 * crlf2lf < infile > outfile
 *
 */

extern struct FileHandle *Input(), *Output();
int Read(), Write();

main()
{
struct FileHandle *infp, *outfp;
char inbuf[1];

  infp = Input();
  outfp = Output();
  while ( Read(infp,inbuf,1L)==1L ) {
    if (inbuf[0]>31 && inbuf[0]<127)
      Write(outfp,inbuf,1L);
    else
      if (inbuf[0]==10 || inbuf[0]==9)
        Write(outfp,inbuf,1L);
  }
  /* no need to Close() any files */
}
```

•AC•

Entertainment Reviews

by Ervin Bobo

SKYFOX
reviewed by Erv Bobo

Of all the games slated to be transferred to the Amiga format, probably none was more eagerly awaited than SKYFOX. It seems that many people who had seen it run on other machines multiplied what they had seen by the power of the Amiga and expected something great. They will not be disappointed.

There are no great changes in SKYFOX, however, but a collection of subtle changes that leads one to the idea that the capabilities of the Amiga finally allowed it to be fine-tuned. And like an automobile newly tuned, it gets off to a roaring start.

I you are familiar with the C64 version, the first thing you will notice is that the theme now sounds like real music, rather than like a clavier. (SKYFOX boasts stereo sound, so be sure your system is hooked up and the volume knob set way up.)

There are the same fifteen scenarios and five skill levels from which to choose, depending on how lucky you feel. Select one and you are whisked to a briefing room where a tactical map shows the location of your own installations, enemy tanks, planes and motherships. Once you have digested this, press the fire button and you will find yourself in the cockpit of SKYFOX, ready to be launched from a long tunnel.

While control can be by either joystick or mouse, I found the joystick to be easiest. Probably because of greater familiarity.

If you picked a multi-level scenario, you can now choose whether to begin your defense at a high level (enemy planes) or a low level (tanks and motherships). Whichever you choose, you will eventually need to do the other in order to beat off the entire attack. The strategy that seems to work for me, is to begin low.

The cockpit is full of instrumentation: a clock that measures elapsed mission time; speed and altitude displays; missile status; a radar screen and much more. A touch of the fire button fires your laser cannon, but you must be lined up on your target. If you can't line up, as when tanks are too close to you, engage a heat-seeking missile by pressing "H" or a guided missile by pressing "G". The missiles have a greater range than your cannon o course, but use them wisely for their numbers are limited.

With all it's controls and instruments, however, SKYFOX is not a flight simulator, nor was it ever meant to be. It is an arcade game, pure and simple and good for hours of mindless entertainment, but with the distinction of being as fast and as furious as the REAL arcade games, the ones that keep asking you for quarters.

Your mission is to fend off an alien attack with as little loss of colonists' life and loss of bases as possible and you are given three planes in which to accomplish this. If you are shot down, you find yourself back looking at the tactical map and ready to go up again.

At any time during the fighting, you may call up the map by pressing Control "R". The map is superimposed over your windscreen. There is also an autopilot. Pressing "A" kicks in your afterburners and sends you to the closest threatened area, dropping you right in the center of a hot situation. However, autopilot will not work if the enemy is near you: clear one area before trying to go to the next.

You should also keep track of your co-ordinates and your course so that you may return to your base for refueling and resupply of ammo. If your base has been destroyed, however, your only option is to stay aloft and keep fighting. You can't win, but you may make the alien victory more costly.

At the end of a round of play, the status board tells you how many colonists and bases are left as well as how many of the enemy have been destroyed.

As to what the Amiga version of SKYFOX offers that is new, consider these: cleaner graphics, more clearly detailed and defined; super sound; improved joystick control; faster loading. And, contributing greatly to the flow of the game, there is no wait for disk access when you fly from a high level to low, or vice versa. This, of course, is because SKYFOX is dumped into ram and as a result everything happens just a bit sooner than before.

However, one thing has not changed: You are still going to have your hands full; things are going to stack up on you; and you will go down in flames more than once. Which brings up another point: even the crash-and-burn routine has been improved.

Enjoy.

ARCTIC FOX
reviewed by Erv Bobo

Like SKYFOX, ARCTIC FOX begs to be enjoyed in full stereo sound and before we are through I will recommend an economical way to do just that. But first...

Aliens have established a base at the South Pole. Firmly entrenched behind a force field, they are busily converting the atmosphere of Earth into a mixture of gases that will be deadly to humans. Because of the hostile terrain, it is thought that the ARCTIC FOX, a supertank of the near future, is the only weapon in our armory that might stand a chance of destroying the aliens and their base.

Once again, you and your trusty computer are off to save the world but this time there is a difference, for you are dealing with the first arcade/strategy game to be constructed especially for the Amiga.

The differences are noted quickly. Key in Level 1, the Enemy Preview, and you will see an enemy tank appear in the distance, heading toward you and slanted slightly to your right, quickly growing in size as he approaches. Then, just short of filling the screen, he slews to the left, now presenting himself in three-quarter profile. You realize you have, for the first time, seen a very close approach to true 3-D. And the redrawing of the tank as it turned to present other faces to you was smoother than anything you have ever seen on a home computer.

And if you are not yet convinced of the 3-D effect, launch a seeing-eye missile and guide its course with the joystick so that it curves around that mountain or arcs over that hill to show you the enemy lurking on the other side.

You see it all in 2-D, of course, for you are looking at a video representation. The heavy armor of the FOX does not allow for windows. To either side of the large video screen, dials and gauges apprise you of your weapons status, oxygen remaining, course, location and elapsed time. In the foreground, hovering over the control panel, is a pair of gloved hands and they belong to you.

You move the joystick and the right hand moves the control lever in the same direction. Use a keyboard command, such as a missile launch and the left hand presses the correct button to carry out the action.

Through it all, you hear the swishing of your treads on the snow and ice; the rumble of your turbine engine; the muffled clanking as an enemy tank approaches; the crack of lightning and the roll of thunder; the whoosh of a missile being launched... The list goes on, but I am still not sure I have heard every sound ARCTIC FOX has to offer.

In Level 1, no play is involved. The enemy vehicle parades before you and a legend below the viewscreen apprises you of the strengths and weaknesses of each piece.

Level 2 is training. Here, the FOX is invulnerable and the enemy weak and stupid. With unlimited weapons, you can destroy everything in front of you.

Level 3 should be called advanced training. The FOX is not invulnerable and the enemy armor is better, but you are still not playing for the money. For that, you go to...

Level 4, the Tournament: here, your weapons are limited; the enemy is strong and crafty and you are going to have your hands full just staying alive - not to mention waiting for the energy flux in the force field so that you can slip through and destroy enemy headquarters.

Due to atmospheric turbulence caused by the alien activities, a thunderstorm or blizzard can blow up at any time and will. Because these storms are electrical, you can almost count on them to scramble you radar, located just below the main viewing screen. If you also happen to be in a blizzard, you are blind except for brief glimpses through the blowing snow. These storms are as well done as everything else in the game and everything else is done very well. This one is going to become a classic.

Documentation is good, setting the scene and explaining the armaments and defenses. A quick-reference card is included and can be kept handy during play. One word of warning: there are tactics available in Level 2 that do not apply in Levels 3 and 4. Since they are silly things like being able to bound into the air, they will not be missed.

The true gamer will appreciate much more the effect of climbing a hill or rolling over the remains of a destroyed tank: in the latter move, one side of your tank will tilt upward and you will hear the motor straining to overcome the obstacle; in the former, your tank tilts up, levels, then tilts down as you descend the hill, adding to the 3-D effect.

Like any good adventure game, ARTIC FOX maintains consistency: if you destroy an enemy at coordiantes X,Y the remains will be there when you return. While the arcade aspects of the game are probably obvious, what with launching missiles and firing cannon, there is a strong element of strategy present as well as a need for navigational skills. You will not master it quickly, nor will you wish to: this is a game to be savored over many sessions.

About the stereo sound: It was impractical to move my main stereo unit from living room to computer room; the portable "boom box" from the family room was, by the family's decree, to stay put. The solution was in the form of a small stereo amplifier from Radio Shack. Mine is the Model SA-150. It takes up little space on the hutch and allows for stereo earphones or speakers or both. Best of all, by feeding the two Amiga sound channels to the Tape input and then using the Y connector to go from Tape output to the Amiga monitor, I can still receive monitor-only sound even when the amplifier is powered down and so have the best of both worlds.

You will enjoy ARCTIC FOX without stereo, of course, but I urge you to try it with it for in this game sound is the fourth dimension.

•AC•

MiAmiga™ File®

a database management system for the Commodore Amiga™.

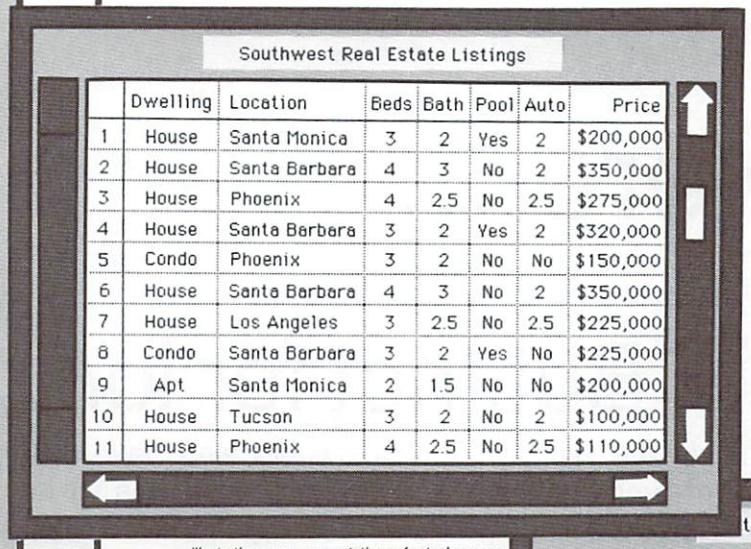
NOW AVAILABLE!

database management system with:

- Pull-Down Menus...
- Written in "C"...
- Vert/Horiz Scrolling...
- Eight (8) Field Types...
- Selection by Example & Range...
- Multi-field, Ascending & Descending Sorting !

SoftWood Company presents

the first serious, professional-quality, full-featured database management system for the Commodore Amiga™.



Southwest Real Estate Listings

	Dwelling	Location	Beds	Bath	Pool	Auto	Price
1	House	Santa Monica	3	2	Yes	2	$200,000
2	House	Santa Barbara	4	3	No	2	$350,000
3	House	Phoenix	4	2.5	No	2.5	$275,000
4	House	Santa Barbara	3	2	Yes	2	$320,000
5	Condo	Phoenix	3	2	No	No	$150,000
6	House	Santa Barbara	4	3	No	2	$350,000
7	House	Los Angeles	3	2.5	No	2.5	$225,000
8	Condo	Santa Barbara	3	2	Yes	No	$225,000
9	Apt	Santa Monica	2	1.5	No	No	$200,000
10	House	Tucson	3	2	No	2	$100,000
11	House	Phoenix	4	2.5	No	2.5	$110,000

Spreadsheet Format provides overview of the database.

Easily define column widths, placement, and justification...

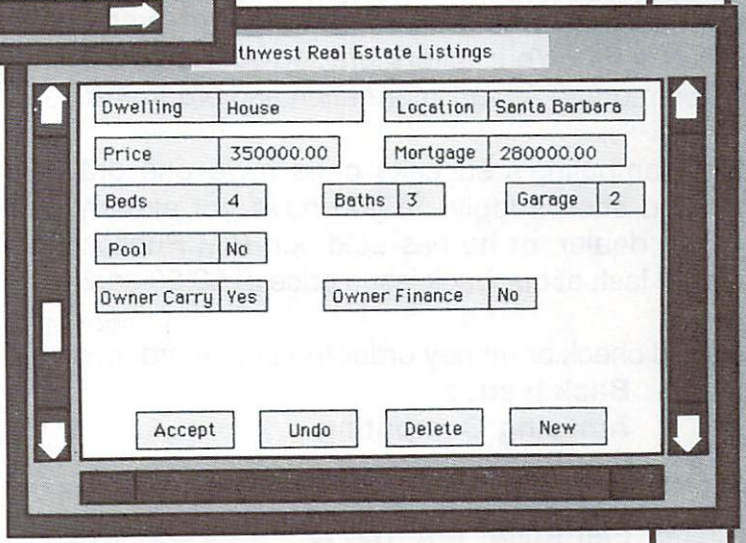
Format numeric fields with commas, dollar signs, and/or decimals...

Transfer quickly between full database and selected records...

Print columnar reports from list including automatic page headings and cummulative totals...

Transfer conveniently from selected record to data entry form.

Illustrations are representations of actual screens.



thwest Real Estate Listings

Dwelling	House	Location	Santa Barbara
Price	350000.00	Mortgage	280000.00
Beds	4	Baths 3	Garage 2
Pool	No		
Owner Carry	Yes	Owner Finance	No

Accept | Undo | Delete | New

Format mailing labels by positioning fields on form...

Automatic scrolling of data within a field during data entry...

Optionally capitalize the first letter of each word automatically...

Modify form as needed for convenient placement of data...

Data entry form automatically created by system during database definition...

Additional features of the New MiAmiga ™ File:

Variable-length record management for optimal space utilization...

Flexible database definition allows user to add/remove fields at any time...

Up to 32,000 records per file, depending upon available RAM and disk space...

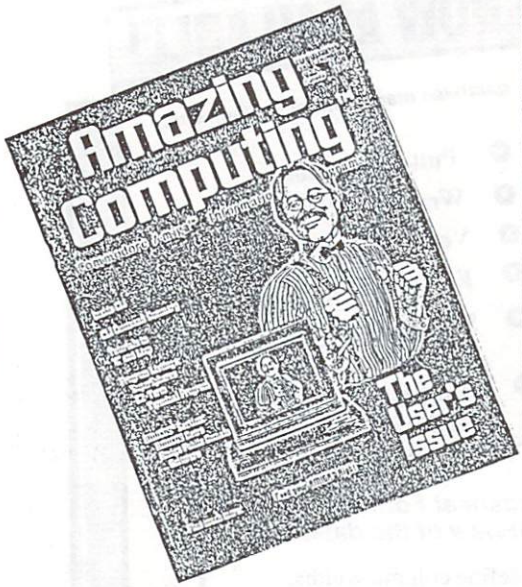
RAM-based file management for fast sorts and searches...

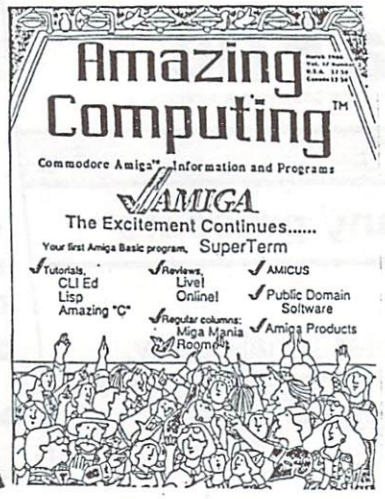
THE NEW MiAmiga File **$99.95** Includes shipping & handling C.O.D. please add $5.

805-966-5884

SoftWood Company
PO Box 2280, Santa Barbara, CA 93120

Dealer Inquiries Welcome. **All Major Credit Cards, Checks, Money Orders, C.O.D. Accepted.**





They're Here....

But, not for long!

PiM Publications, Inc. wishes to thank all of the fantastic Amiga owners who have accepted Amazing Computing as a resource for their Commodore Amiga. The response was far beyond our expectations. Your letters praised Amazing Computing™ and encouraged us to continue to deliver Commodore Amiga information and programs.

Amazing Computing's success does have one draw back, our First three issues are in great demand and short supply. If you have not already purchased a copy of these treasures from your Amiga dealer, or he has sold out, PiM Publications will make back issues available, for as long as they last, at our Back issue price of $3.50 each.

Please send check or money order (no phone orders please) to:

Back Issues
Amazing Computing
PiM Publications, Inc.
P.O. Box 869
Fall River, MA 02722

If you want to be sure you do not miss an issue, then subscribe!
Amazing Computing is available by subscription, (no phone orders, please) for 12 issues:

$24.00 in the United States
$30.00 in Canada and Mexico
$35.00 Overseas

Amazing Computing™ your resource to the Commodore Amiga

"Roomers"

by The Amigo

It's not easy to write about the cutting edge of product announcements when you have a lead time of more than a day (!), but some roomers are worth writing about anyway. The "Amiga re-launch" is an example.

Soon after CBM got its financial woes settled, it went ahead with big plans to "re-launch" the Amiga. That means giving the monitor away with the system, stepping up advertising, making a big splash at Comdex, and introducing some significant products.

The "buy an Amiga get a monitor free" is scheduled to run until May 15. Come on. Who lowers prices and then raises them? Look for CBM to announce that "the promotion was so popular, we're extending it another two months" or something similar.

CBM is now shipping the Transformer, with 5-1/4" drive. The product isn't as complete as CBM wants it to be (some problems with color screens and 40-column setups), but I'm told no further work will be done on it, because it's late enough already, and there's this other thing...

Commodore is supposed to have a big showing at Comdex/Spring, April 28 - May 1. Now of course by the time you read this, it will be after Comdex. CBM is expected to have from 50 to 75 Amigas set up, filled with products from both CBM and third party developers. The latest news I have is that Commodore will announce an add-on side car to the Amiga that will have either an 8088 or 80286, 512K (why not 640K?) memory, a color graphics adapter, a 5-1/4" floppy drive, room for an IBM hard disk and three expansion slots for $500. I guess they're going to rip out the insides of their PC-10 and PC-20 IBM clones that have done so well in Canada and Europe and put it in a box to hang off the Amiga. Sources say your dealer should have one to show you by the end of this month.

The folks at Commodore-Amiga are considering a version of Preferences that will allow you to bring up the Workbench screen in 640x400 (interlaced) mode. It seems that many people are looking for this! The problem, of course, is flicker of single-width horizontal lines, especially the drag bar. Version 1.2 of the system will see a different style of drag bar to relieve this. Future versions may see a different default text font, similar to what Sun did to reduce flicker on their interlaced workstation. Also in version1.2: faster directory listings, an extra screen in preferences to let you set the serial device characteristics (including MIDI speed), new library calls, and some bug fixes. There are other toys out there, like FED (Font EDitor), DISKDOCTOR, ADDBUFFERS (to your drive), and MicroEmacs (uses Intuition, menus, mouse, supports interlace mode,loadable function keys, etc). I don't know if these programs will see the 1.2 distribution, because CBM is talking about a "programmer's utility" disk. I've also heard about a 'best of the public domain' disk from CBM ... these two disks could be the same.

Some rumors flying around about the EXTRA_HALFBRIGHT mode (where you can get 64 colors at once). As you may know, the first umpteen-thousand Amigas were shipped with revision 5 of Denise, which does not have the EXTRA HALFBRIGHT mode, while the rest were shipped with rev 6, which has it. It is not true rev 7 took away EXTRA HALFBRIGHT ... it's in there! Amigas are now being manufactured with rev 8 of Denise,and the EXTRA HALFBRIGHT mode is still a feature. Still no word on how CBM will make new Denise chips available to people without the half-intensity mode, but the chip is socketed, so there is no soldering to be done.

For you Amiga C programmers, an undocumented option on ALINK will decrease your link times as much as 80%! The option is 'faster', which should appear at the end of the ALINK command line. Although it speeds up the linker, it also does no memory checking, nor does it deallocate memory as it should. It was put in as a quick hack by a staffer at Amiga who was sick of waiting for ALINK. It was not supposed to find its way into the release version, but here it is. Use it for speed, but watch your memory usage!

The Amiga music scene is a mess. First, Cherry Lane goes out of business. They were making a MIDI interface box and some software for the Amiga. The program 'Concertcraft' was a subset of Harmony, a Cherry Lane product, so it was also axed. Next, Commodore decides to get out of the MIDI business altogether, even though they had a boatload of MIDI interfaces already assembled. Then they decide maybe they should at least sell the MIDI interface, since they said they would and have made them already, but nobody will see the interface for a few months! Hayden Software, who announced Sound Vision, decided they didn't want to develop for the Amiga after all. Poof. Instant Music,

from Electronic Arts, has been scrapped. Deluxe Music Construction Set is due "sometime this summer". The rack-mountable Amiga reportedly died along with Cherry Lane.

Oh well. Some third party developers stand to make a TON of money, from my viewpoint. The Mimetics packages should do just about everything you need to do with a MIDI package; at least to start. The box from Golden Hawk Technology is cheaper and has more features than the CBM MIDI interface would have (MIDI in, 2 outs & drum sync; as opposed to MIDI in/out/thru), as well as being available sooner.

Why all this suddenness about the MIDI software and the IBM packages? I hear that there was a bit of a shakeup at CBM, the MIDI and IBM products were coming along far too slowly for the upper management's tastes.

Something is shaking in the developer support area. Many developers have complained the support they have been getting from West Chester has not been enough. Carl Sassenrath, designer of the Amiga's Exec, has asked publicly for comments on Amiga support, and he promises changes are in the making. I wish I knew more. My guess is Carl (who no longer works for Commodore-Amiga) is planning to start an Amiga technical support service. Others, who I have talked to and seem to know more agree, things will be looking up for developers, and Carl will be one reason.

On the subject of new product directions... Amiga has completed its ZORRO expansion specification. Although the Amiga provides an 86-pin connector, the ZORRO spec calls for 100-pin boards! These extra 14 pins will be used for intelligent boards to talk to each other without the Amiga even knowing they are doing so. I'm sure we will see more innovative uses for those extra 14 pins!

Also in the future: after version 1.3 of the system, which should be the release where C-A feels the bugs are worked out, Kickstart will be put into ROM on all new machines. AmigaDOS/Workbench will still come on floppy, so future improvements will be possible.

In the premiere issue of Amazing Computing, I reported on the Ranger, the new Amiga. Well, yes and no. The latest information I have is that "Ranger" is an ideology, not a machine. It describes a range of products. The current Amiga will become a new Amiga by swapping chips, boards, and external devices. You decide what configuration you need ... 1024x768 resolution? New graphics chips, new monitor,etc. More number crunching? 68020 at 14mHz and a math coprocessor. Better sound? A new 16-bit Digital-to-Analog converter is on the way. IBM compatibilty? No Problem. UNIX? Got that coming too. Commodore is looking at the Amiga as a three-in-one machine (AmigaDOS, MS-DOS, UNIX) like they did with the C-128 (C-128, C-64, CP/M).

Having said that, there IS a new Amiga about to be announced. You should see it before August, and it will be similar to what is described above, all in one package. Price around $4000, available by Christmas. It will be positioned as a UNIX-based CAD workstation.

Also heard:
an Amiga A1000 with expansion slots built in. It's a little thicker than the current Amiga. Don't expect to see this one though, as the general theory is that it will create too much confusion for the consumer.

Other hardware notes: The ZORRO backplanes will be available from DSI, Byte-by-Byte and Ameristar. Ameristar has announced an Ethernet interface and NFS, the Sun Network File System, which allows transparent file serving across an Ethernet. Hook your Amiga to a VAX through NFS and never worry about file storage again! Ameristar also has prototyping boards available now.

Lastly, there's a hot rumor that Commodore-Amiga has developed a DMA-driven 20MB hard disk and may show it at Comdex.

Miscellaneous notes ...

Look for high-end music and video manufacturer Fairlight to incorporate the Amiga into future products ...

TeX, a text processing package designed by Donald Knuth at Stanford, has been ported from Pascal to C and compiled on the Amiga. Look for it this summer ...

TextCraft has been revamped. It now better supports multitasking, uses Intuition for its user interface, and has some new features. It will be called TextCraft Plus and will be available this month ...

Sublogic has delayed release of Flight Simulator II and Jet until July and August, respectively ...

SSI, makers of Word Perfect, are porting it to the Amiga ...

AutoCAD is being ported to the Amiga, to compete with Aegis' Pro Draw CAD package ...

Microsoft is updating AmigaBASIC, and is working on more products for the Amiga ...

See you next month!

•AC•

Build Your Own IBM Drive connection

by E.P. Viveiros

Amazing Computing Hardware Editor
with assistance from Jim Rutherford
and John Foust

The Amiga computer derived its name from the Spanish word amigo, meaning friend. We are about to see how friendly the Amiga really is as we connect an IBM™ compatible 5 1/4" disk drive to the Amiga disk drive interfacing hardware.

Interfacing an IBM compatible 5 1/4" disk drive to the Amiga is a fairly straight forward task. Nearly all of the required interfacing circuitry has been designed into the Amiga's disk controller with the exception of one unique function.

The Amiga disk drives and controller are unique, they latch the motor drive signal at the time its' SELn turns on and remains latched until the next SELn. The drives are also designed to turn off after a system reset. The circuitry required to emulate this unique Amiga function consists primarily of a latching device (in this case a 74LS74 flip flop), a line driver, 74LS05 and several pull up resistors.

All of the parts required to construct the "latch board" are readily available, with the exception of the DB23 connector, required to attach the adaptor to the rear of the 3 1/2" disk drive. I know of no readily available source for the required DB23 connector and unless you are looking for a fairly large telephone bill, don't start making phone calls; it seems as though I called a thousand different houses and no luck.

There are a number of ways in which any hardware project can be brought into reality; but only two reliable methods will be considered in this article.

The first, and in my opinion the preferred method, is through the use of a Printed Wiring Board assembly. A printed wiring board , however is not easily fabricated, especially when one is dealing with a two sided PWB.

For those readers not willing to design and etch the required PWB themselves, wire wrapping is the way to go. I have provided three forms of documentation for the latch assembly:

1. A PARTS LIST
2. A SCHEMATIC
3. A FROM TO CABLE CONNECTION CHART

I will not go into the details of how to wrap the latch PWB, build the cable or any detail here, since no one should attempt this project unless they possess the basic electronic know-how. The enclosed lists and diagrams should be clear for the technically competent. I am not saying it is a difficult project, in fact it is quite easy, just be sure you know what you are doing and you will not be sorry later.

The latch assembly can be wired using the information on the schematic or the From-to list. One word of CAUTION .....The Amiga J7 connector has pin outs for both 5 and 12 VDC but they must not be used to power the 5 1/4" disk directly... According to the Amiga hardware manual, the output of the pins are 270 and 160 milliamps, while the 5 1/4" disk will require approximately 1 ampere. However, J7-12 can be used to power the latch board since the current drawn is minimal.

They say, ..."THE PROOF OF THE PUDDING IS IN THE EATING."' ..... The 5 1/4" disk coupled with Commodore's Transformer software opens up a whole world of available software. Thousands of programs are available and most will run under the transformer software.

This hardware project may not be the most elegant way of solving the Amiga Disk Motor Latching phenomenon, but it works. I know, since I have a SHUGART SA455 wired in and running. Good luck to anyone attempting this project. .....TAKE YOUR TIME.....CHECK YOUR WORK OUT... AND IF YOU FOLLOWED INSTRUCTIONS PRECISELY, YOU WILL HAVE YOUR SA455 UP AND RUNNING IN NO TIME AT ALL.....

HAPPY COMPUTING
E.P. Viveiros

Hardware Editor,
Amazing Computing Magazine

For information on boards and kits to build your own IBM Connection, contact:
Teddy Bear Systems
C/O Amazing Computing, PiM Publications Inc.
P.O. Box 869
Fall River, MA 02722

Schematic Latch Assembly

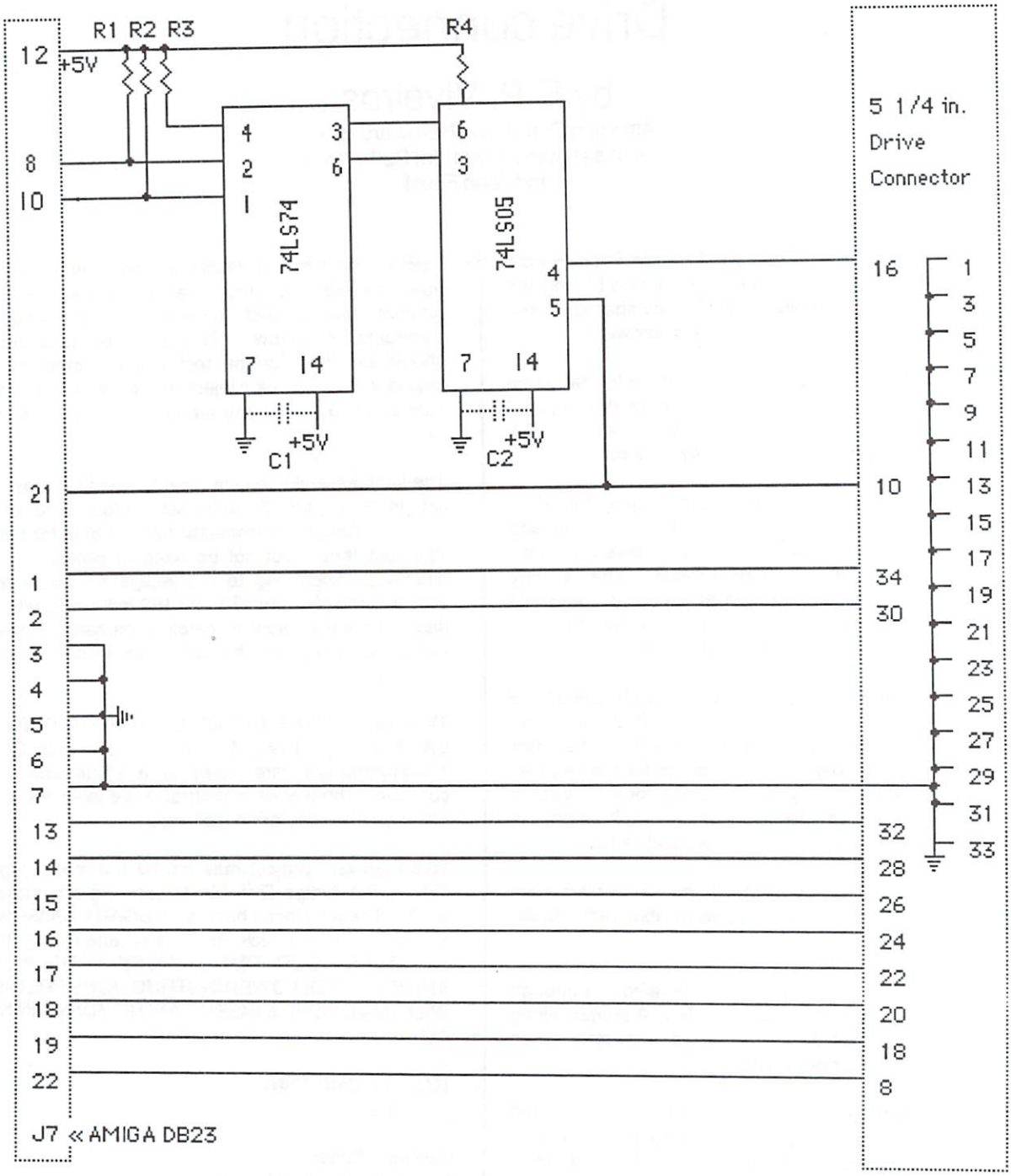
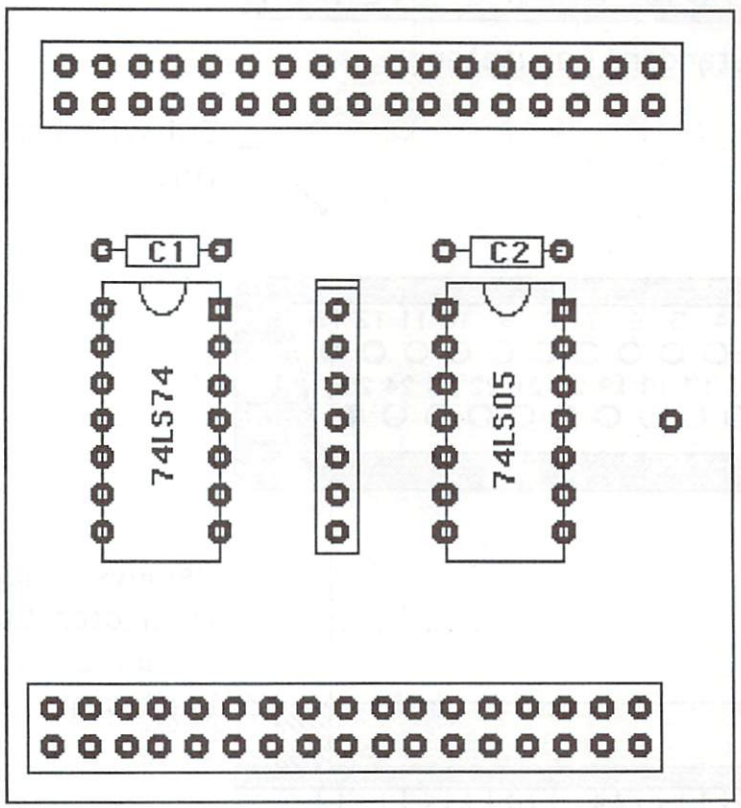
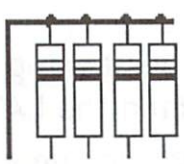


Figure 1

The board design is strictly a matter of personal taste and the following are simple suggestions or approaches. The boards presented are intended to be attached between the Amiga and the 5 1/4 disk drive, but could as easily fit into the drive. The choice is yours. Amazing Computing claims no liability for this assembly, only that it works for us.





Alternate for SIP package.

Figure 2

DB25/DB23 CONVERSION

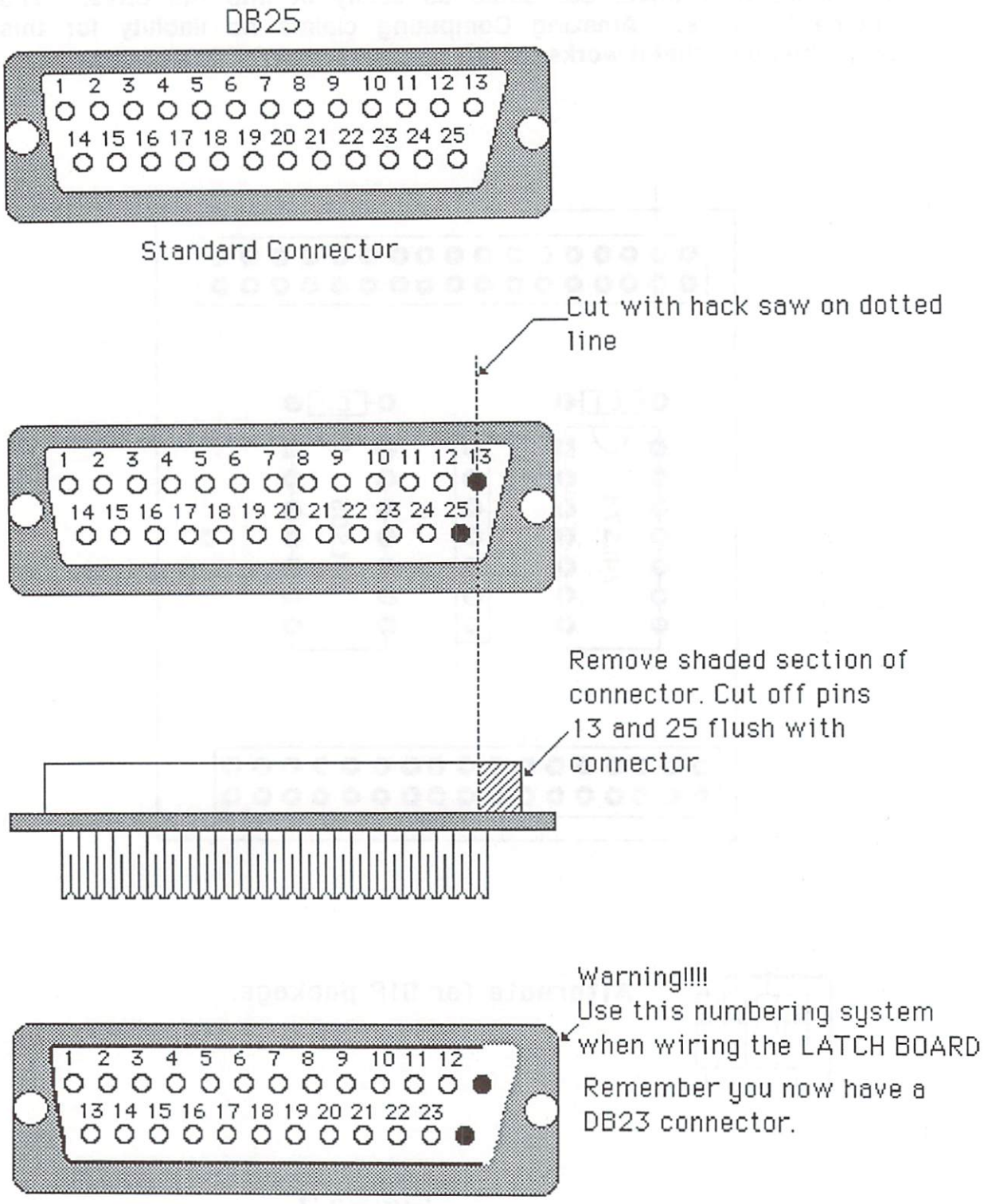


Figure 3

Cable connections

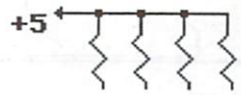
FROM	TO	COMMENTS
J7-22	Shugart-8	
-19	-18	
-18	-20	
-17	-22	
-16	-24	
-15	-26	
-14	-28	
-13	-32	
- 7	Shugart-	Wire J7-3,4,5,6,7
- 6		to all odd pins on
- 5		Shugart connector
- 4		
- 3		
- 2	-30	
- 1	-34	
-21	-10	
SHUGART-10	74LJ05-5	
74LS05-4	Shugart-16	
74LS74-3	74LS05-6	
74LS74-6	74LS05-3	
J7-10	74LS74-1	
74LS74-1	R2	NOTE R1,R2,R3,R4
J7-8	74LS74-2	have common end
74LS74-2	R1	tied to +5vdc (J7-12)
R3	74LS74-4	
R4	74LS05-6	
74LS74-7	74LS06-7	
74LS74-7	J7-3	Ground
74LS74-14	74LS05-14	
74LS74-14	J7-12	+5vdc DO NOT POWER
C1-1	74LS74-7	51/4 DISK FROM THIS
C1-2	74LS74-14	PIN OR J7-23(+12vdc)
C2-1	74LS05-7	
C2-2	74LS05-14	

Figure 4

SA455

Shugart/Amiga Jumper Configuration

The jumpers on the drive must be in the indicated positions for the LATCH BOARD to function properly.

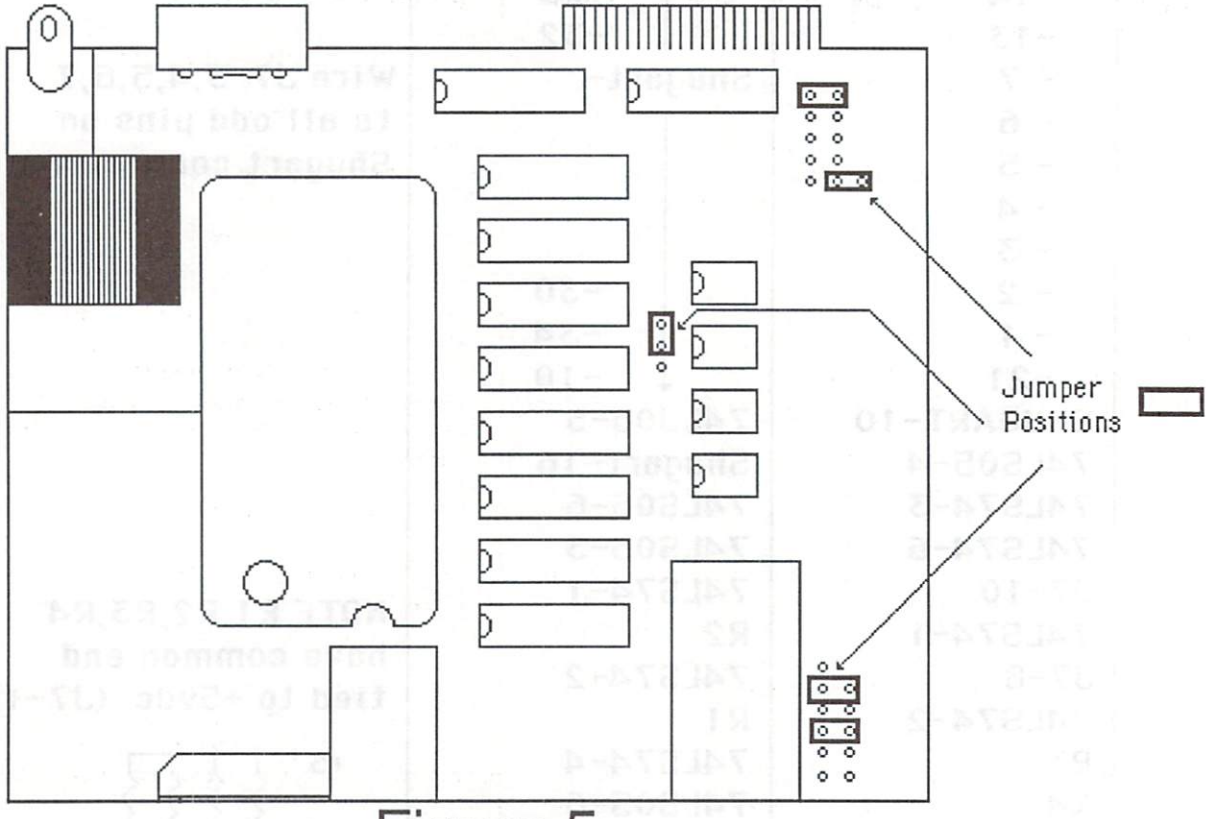


Figure 5

Latch Assembly Parts List

1- 74LS74
1- 74LS05
4- 2200 Ω 1/4 Watt Resistor
(A SIP would be an improvement)
2- .01 mf Decoupling capacitor
2- 14 pin wire wrap sockets
Mounting board with .100 hole spacing
1- DB25 male connector (DB23 see instructions)

1- 34 pin edge connector for disk drive
1- Power supply +5vdc,12vdc 1 amp minimum

1- Shugart SA455 disk drive
Connectors and cabling (dependent upon layout)

None of the above parts are particularly critical so feel free to substitute; just do not get carried away. Any variety of the 7474 or 7405 will work, but be cautious of the 54 series since most have different pinouts. The two capacitors are not necessary to the board's function but are included because they are good design practice. As for the resistors or sip network, just about any value resistance and wattage will do.

I have only had time to experiment with Shugart disk drives, however other manufactures should work as well also; remember I said should and I make no guarantee! •AC•

Forth Part 2

by Jon R. Bryan

As of this writing, I still have not received a working implementation of Forth for my Amiga, even though one is available in the public domain. It is probably for the best, since I do not want to burden you with too many details yet. I would like to spend a bit more time presenting a general overview of the structure and philosophy of Forth before getting into details specific to the Amiga.

If you are interested in learning Forth or even if you have extensive experience with the language, I recommend that you buy a copy of the book "Thinking Forth" by Leo Brodie. It is published by Prentice-Hall, and any bookstore can order a copy for you if they do not have it in stock. It is subtitled "A Language and Philosophy for Solving Problems," and gives the best exposition of Forth style that you are likely to see. It has certainly given me a number of useful ideas which I have been able to use (and not just in Forth programs), and it is a joy to read. It is one of the most clearly written books I have ever read on any subject.

THE ELEMENTS OF STYLE

Forth has a number of unusual features. First, it is interactive. You can execute words and compile definitions directly from the keyboard. Second, it is a "stack-oriented" language. You may not consider that a "feature," but the approach returns a number of benefits which I will be discussing in much greater detail. Third, Forth uses postfix notation, which you may have heard referred to as "Reverse Polish" notation, or RPN. The use of RPN is largely a consequence of the use of a stack. And last (for now), Forth is "extensible." Writing a program in Forth consists of extending the language to encompass an application. If you do not understand all that, bear with me. It will eventually become clear.

Because it is interactive, and because by the time this tutorial is printed there should be several versions of Forth available for the Amiga, you might want to try the examples as you go. The best way to learn Forth is by using it.

One of the first things you have to learn in Forth is how to handle the "stack." In computer jargon, a stack is a "Last In, First Out," or LIFO, data structure, and is often compared to the spring-loaded stacks of plates and trays you see in cafeterias. Forth's data stack provides a simple mechanism for passing parameters between words, and also for hiding information about those parameters.

Passing parameters and calling functions is implicit in Forth. In some languages you might be required to use the expression:

```
CALL DRAW_LINE(Xstart,Ystart,Xend,Yend)
```

explicitly calling out the variables to be passed to the function DRAW_LINE, whereas in Forth you would simply say

```
DRAW_LINE
```

and hide the details about which parameters DRAW_LINE expects and how they get on the stack. No CALL or its equivalent is necessary, since DRAW_LINE "knows" how to call itself, and where to get its parameters. It may expect Cartesian coordinates, but they could just as easily be Polar, and at this level you do not care. In fact, in line with much recent thinking in computer science, **YOU DON'T WANT TO KNOW.** The more information about parameters, data structures and algorithms you hide, the smaller the chances are for bugs and design errors to creep into your programs, and the easier it is to make changes later.

So implicitness is great, but how do you use the stack? It turns out to be very simple -- and implicit. Since Forth is interactive, if you want to push a value onto the stack you can simply type in the value and hit RETURN. Pushing more than one value onto the stack is as simple as this (and you can try it now):

```
1 2 3 4 5 <RETURN> OK
```

The numbers 1 through 5 are now on the stack, with 5 on top. Once Forth is finished doing what you have asked of it, it always concludes by saying "OK."

Forth provides another word for printing out the top stack value, called . (dot). Executing . pops the top stack value and outputs it to the screen. Here is another example which will type out all the values we just placed on the stack:

```
. . . . . <RETURN> 5 4 3 2 1 OK
```

Executing another . would result in an "Empty Stack" message unless you would put something else on the stack previously.

Compiling a number into a definition is just as simple. If you needed a word which would put the values 1 through 5 on the stack, here is how you could do it:

```
: 1thru5 ( -- 1 2 3 4 5 ) 1 2 3 4 5 ;
```

and when you executed 1thru5, the values would be pushed onto the stack.

The basic stack operators are **DUP** , **OVER** , **SWAP** , **ROT** and **DROP** . The word DUP duplicates the top value

on the stack. OVER makes a copy of the second value, leaving the copy as the new top value. SWAP exchanges the positions of the two top values. ROT moves the third value to the top. DROP pops the top value and discards it. Here are examples, where the value on the top of the stack is the furthest to the right.

Stack Before		Stack After
1 2 3	DUP	1 2 3 3
1 2 3	OVER	1 2 3 2
1 2 3	SWAP	1 3 2
1 2 3	ROT	2 3 1
1 2 3	DROP	1 2

I'll save two other stack operators called PICK and ROLL for later discussion.

One natural result of the use of a stack is postfix notation. If you want to add two numbers they first have to be on the stack.

```
2 2 + <RETURN> OK
. <RETURN> 4 OK
or
2 2 + . <RETURN> 4 OK
```

Another Forth convention is operators destroy their arguments. In other words, when you execute 2 2 + , the two values will be replaced by the single result. Likewise with subtraction, multiplication and division. Postfix notation is also used for logical tests, which include < (less than), > (greater than) and = . (For some reason I had trouble in the beginning remembering in which order < and > worked on their arguments, until I finally realized that 3 4 < is the same as 3<4.)

Forth includes a full set of conditionals for writing structured programs. These include IF ... ELSE ... THEN , REPEAT ... UNTIL , BEGIN ... WHILE ... REPEAT and DO ... LOOP . Postfix also applies to them, as in one of the examples from last month:

```
: ?FLOSS  LAZY? NOT IF FLOSS THEN ;
```

The flag for IF is left on the stack by the two words LAZY? NOT. If the flag is "true," FLOSS will be executed, otherwise the word will do nothing. In other words, if you're not lazy, floss your teeth.

The use of the stack and postfix notation imposes an extra burden on you, the programmer, but it also provides many benefits, some of which I have attempted to illustrate here. More will become apparent as you use the language. Well-written and designed Forth programs can be very easy to read; partly because algorithms can be expressed so concisely, without the clutter of CALLs, parameter lists, or the other excess baggage imposed by most other languages.

The final feature of Forth which I would like to cover is its extensibility. It is one of the language's greatest strengths, and one which is shared by very few of its contemporaries. All of Forth's words are contained in an open-ended "dictionary." Writing a program consists of defining new words in terms of existing words and creating new machine-level words when necessary. They all become a part of Forth as it exists on that particular machine.

A program in Forth is simply a word which encompasses an entire application. That word is composed of other words, and the chain continues down to the lowest machine-level primitives. Even the primitives are words, and they look and behave just like the so-called "high-level" words. One of the central concepts of Forth is that all code should look and feel the same, whether it's "low" or "high" level. Superficially you can't tell whether a word has been written in machine language or Forth.

All of these features, plus many more, combine to make Forth a very elegant, powerful language. I am eagerly awaiting Multi-Forth from Creative Solutions, the people who produce MacForth for the Macintosh and versions of Forth for several other 68000-based computers. They have told me that their implementation for the Amiga is the best one they have done so far, and that it will begin shipping on the 31st of March (next week, at the time I am writing this). UBZ Forth was supposed to be here already, and I will have the public domain Mountain View Press implementation in my hands this week.

So, next month, I'll be writing about Forth on the Amiga!

•AC•

New Amiga Products From The Developers of Amiga C.

Amiga C Compiler—$149.95
Everything you need to develop programs on the Amiga, including a full set of libraries, header files, an object module disassembler, and sample C programs.

Unicalc—$79.95 A complete spread sheet package for Amiga, with the powerful features made popular by programs such as VisiCalc, SuperCalc, and Lotus 1-2-3. Unicalc provides many display options and generates printed reports in a variety of formats and print image files. Supports 8192 rows of 256 columns, and includes complete on-line help.

Lattice MacLibrary—$100.00
The Lattice MacLibrary is a collection of more than sixty C functions enabling you to rapidly convert your Macintosh programs to run on the Amiga. this allows you to quickly and efficiently take advantage of the powerful capabilities of the Amiga.

Lattice Make Utility—$125.00
Automated product generation utility for Amiga, similar to UNIX Make, LMK rebuilds complex programs with a single command. Specify the relationships of the pieces, and automatically rebuild your system the same way every time.

Text Utilities—$75.00 Eight software tools for managing text files. *GREP* searches for specified character strings; *DIFF* compares files; *EXTRACT* creates a list of files to be extracted from the current directory; *BUILD* creates new files from a batch list; *WC* displays a character count and a checksum of a specified file; *ED* is a line editor which utilizes output from other Text Utilities; *SPLAT* is a search and replace function; and *FILES* lists, copies, erases or removes files or entire directory structures.

Lattice Screen Editor (LSE)—$100.00 Fast, flexible and easy to learn editor designed specifically for programmers. LSE's multi-window environment provides the editor functions such as block moves, pattern searches, and "cut and paste". Plus programmer features such as an error tracking mode and three assembly language input modes.

OTHER AMIGA PRODUCTS AVAILABLE FROM LATTICE:

Panel: Screen Layout Utilities—$195.00
Cross Compiler:
MS-DOS to Amiga C—$250.00
dBC III:
library of data base functions—$150.00
Cross Reference Generator—$45.00

With Lattice products you get *Lattice Service* including telephone support, notice of new products and enhancements, and a money-back guarantee. Corporate license agreements available.

Lattice

Phone (312) 858-7950 TWX 910-291-2190
INTERNATIONAL SALES OFFICES:
Benelux: De Vooght. Phone (32)-2-720-91-28. England: Roundhill. Phone (0672) 54675
Japan: Lifeboat Inc. Phone (03) 293-4711 France: SFL. Phone (1) 46-66-11-55

The Amazing Mail:

Dear Amazing Computing Editor:

I have recently bought an Okimate 20 printer for my Amiga. Some of my homemade Basic programs generate a color picture, however, I cannot figure out how to get my Amiga to print the output window picture on my Okimate. I know this involves some CLI manipulation . . . but what?? Would you please ask someone to write an article on how to make a hard copy of a color Basic output window on the Okimate (and other) printer(s). I know this information would be very useful and valuable to your A.C. readers.

Love your magazine. Looking forward to the next issue.

Sincerely,
Stu Thomson
Chelsea, MA

Mr. Thomson,
Having no ready answer, I am placing your request in this forum with the hope that one of our readers will chip in. Ed.

Dear Sirs,

Enclosed please find a one year subscription to your incredibly informative publication Amazing Computing. I constantly await the arrival of the next issue at the local Amiga dealer and I am sure he will appreciate me not coming in every two days to see if it has arrived.

Thank You,
Tim Levin
Topeka, KS

Keep visiting your dealer, you do not want to miss all the new Amiga products on the way.

Dear Friends,

I just looked at the March issue of your wonderful journal. It is great. To get this I had to leave an arm with the person I borrowed it from. So HELP! Please start my subscription.

Thank you,
"Lefty"
Amberse Banks
Modesto, CA

Please be more cautious. Back issues are still available and you would not have needed to sacrifice your appendages.

Dear Mrs. Gamble,

I most definitely want to feed my Amiga right, lord knows it needs it. Please accept my CHARTER SUBSCRIPTION to your new publication "AMAZING COMPUTING". I have a copy of your premiere edition, but being a beginner, I really can't afford to miss an issue of your valuable publication.

Sincerely,
Patrick E. Dugan
McFarland, WI

Dear Amazing of Amazing,

I'm sorry I missed the premiere issue and I hope I'm not too late to get one.

One of the reasons I bought an Amiga is that Borland was going to come out with Turbo Pascal for the machine. I have given up on them ever coming out with what they promised over six months ago. If you guys know something that's even the least bit encouraging please let the world know about it.

I recently purchased a copy of COMPUTES!'s AmigaDOS Reference Guide by Sheldon Leeman and Arlan Levitan. I think it is a terrific book and I recommend it to all. They do a great job in explaining the DOS commands and providing examples and the built-in editors are covered in great detail.

Don't you think the drives on the Amiga are slow? Do you think they planned it that way so everyone would run right out and buy a hard drive? I think my not-souped-up Kaypro II will beat the Amiga in transferring files. If I get time I'll do a test and let you know.

Enough already, I gotta get back to my Amazing Computing and my amazing Amiga. I think you have a value packed product even though it has the unlikely title of Amazing Computing.

Sincerely yours,
Benjamin T. Bajorek
Dearborn, MI

Unlikly Title? If the Amiga is amazing, why shouldn't computing on it be as thrilling?

Hi Don, et. al.
I was very pleased with your first two issues. Judging from them AC will be an invaluable resource for all Amiga owners.

I am also pleased that CompuServe and the Amiga Forum have been able to play some role in the formation of this great magazine. I hope that all concerned groups can continue in supporting each other in supporting the Amiga (Amicus, A.C. CompuServe, BiX Use Net . . .)

Looking forward to many more great AC issues.

George M. Jones/Amiga Forum
Worthington, OH

George, we like Compuserve also, but don't forget Peoplenet as explained by John Foust in this issue's AMICUS column

Gentlemen,

I saw, I read, I want!

I would like to subscribe to Amazing Computing. Hope to hear from you soon.

Sincerely,
Steve G. Mankowski

We heard, we listened, you got!

Mr. Hicks,

Thank you for taking the time to help me out with my "beginner's problems" related to using Amicus Disk #1.

After our conversation, I purchased the DOS- Manual from Bantam. This lead me to try from ABasic to load ABasic stuff/Games/Saucer.bas. The program loaded but would not run. As you suggested, I went to CLI and used ED to delete line 0.

The program then worked like a charm.

I had been a fan and subscriber of your magazine, now I'm also a fan of the Managing Editor!

Sincerely,
Harlan Tait
West Haven, CT

Thanks. However, I got all my information from George Musser's first two articles on CLI in our first two issues. The credit belongs to George.

Dear Sirs:

I recently entered a program called Racetrack, from a book "Programming Languages:featuring the IBM PC and compatibles" by Stiegler and Hansen.

It contains a structure with which I am not familiar. (ie. A=B=C) I'm getting an "illegal function call" in the requester box.

If anyone knows of another way to express this, I'd appreciate knowing of it. I'd like to get by this stopping point.

The code as written is:
28530 IF ONMAP THEN HITFENCE = (MID$(MAP$(YNOW%),XNOW%,1) = FENCE$):
FINISHED = (MID$(MAP$(YNOW%),XNOW%,1) = FINISHLINE$

Also, when a line such as

print tab (30-I); (I,J);

prints down the screen and you have to remove tab to get it to print across, I do get a bit exasperated.

I like your magazine but hope that you will give a review to any programming books that come out. I'm still mixed up on proper use of conditionals.

Sincerely,
Allan W. Wardell, O.D.
Providence, Ri

Working.....

Dear Mr. Hicks:

You must imagine how excited I was to find your magazine Amazing Computing at the MicroLimits store up in Smithfield. With the incredible lack of documentation on the Amiga (even from Commodore) I have to say that finding your magazine made my week. But finding out that the publisher was practically just up the street, drove me absolutely wild!

Regards.
Paul White
Tiverton, RI

We are a small and unassuming lot.

Dear Readers,

Please continue to send in your thoughts and suggestions. We do love hearing from you.

The AC staff.

•AC•

The Amazing C Tutorial:
part four

Manx Aztec C compiler system review

by John Foust

This column is both a review of Manx Sotfware System's Aztec C compiler and the fourth installment in the C tutorial series.

A review of a compiler can give any C programmer a better insight into the language, and it may answer questions in the minds of budding programmers.

The review must start with an apology to both Lattice and Manx, since this review draws comparisons between the two compilers. That in itself is a normal function of a review, but Amazing Computing has yet to review the Lattice compiler, the official development system for the Amiga. It has been available to developers for over a year.

The Manx compiler has demonstrated speed and efficiency in code side and execution time, and draws much attention for these differences, so the two development systems are compared and contrasted here.

Manx Software Systems offers the Aztec C compiler in two varieties: Developer and Commercial. The list price of the Developer system is $299, and the Commercial system is $499.

Some software ads have mentioned a Personal version, but this is currently a non-product - it does not exist. It may be sold in the future, according to a Manx representative. The personal version was advertised as a less-optimized compiler, provided without an assembler, and with fewer library and utility routines.

A less-optimized compiler produces slower and larger programs, in general. A developer would prefer an optimized compiler, since the redundant machine language instructions would tend to bloat and slow any program.

An optimized compiler applies a set of rules to remove excess or redundant machine language instructions. An example of optimization is to keep often-used values in registers, since register instructions execute faster than memory-to-memory instructions, as a rule.

According to people who have called Manx to order the compiler, there have been several special prices for the package. One person mentioned a $100 discount given to Lattice owners, and another said they got a 25% student discount.

The Developer package includes the C compiler, assembler and linker, the necessary object libraries and an object librarian.

The Commercial package includes the source code to the Aztec library functions, and a series of utilities, including a text editor, detailed below. These include the programs in the Developer package and more object file conversion utilities.

Upgrades

The Commercial system also comes with a year of free updates. The Developer system does not. With the developer system, updates are $15 for minor updates, and as much as $50 for a major version number upgrade.

After a few weeks, my first upgrade package arrived, since I had the commercial version. It was a set of pages to replace and augment parts of the manual.

The Aztec compiler converts C source code to human-readable assembly language source code, and then assembles that code to produce the object file.

AmigaBasic Tips

by Rick Wirch

This tutorial was written to serve a twofold purpose, one, to teach people how to use some powerful graphics routines not directly supported in AmigaBasic, and two, to help people who have used ABasiC, convert their existing programs to AmigaBasic.

For those of you who have used both BASIC's obviously AmigaBasic is so much easier to use, and has so many features that ABasiC looks pale in comparison. But, many people who have used ABasiC may miss some of the features that Microsoft chose not to implement.

These new features are implemented using one of AmigaBasic's most powerful abilities, the ability to call functions written in other languages. (These are all functions in the graphics ROM routines, or KICKSTART RAM to be more precise).

To use these routines you must first have the 'graphics.bmap' file in the directory (or drawer) where your AmigaBasic program is going to reside. You may either drag the 'graphics.bmap' icon into the drawer on your work disk from the 'BasicDemos' drawer on your 'Extras' disk, or write your program in the 'BasicDemos' drawer where the 'graphics.bmap' file is presently.

Second, at the top of the program you need to have a statement like this: LIBRARY "graphics.library".

Third, if you have any DEFING, DEFSTR, DEFDBL, or DEFSNG statements that have the same letter range as the first letter of one of the following functions, the function must be called explicitly as a function that returns a long integer. For example, the statement: CALL Move(RP&, X, Y) would become: CALL Move&(RP&, X, Y) if there was a DEFINT M-Z statement.

Fourth, all of the graphics routines need a pointer to the RASTPORT. This pointer can be initialized with the statement:

RP& = WINDOW(8)

after any custom windows are opened.

Finally, upper or lower case letters are important in the name of the routine. For example, Move would work, but MoVe would not work.

With this, all of the initialization necessary to call graphics routines has been completed and we are ready to call the routines. I will now give a brief explanation of some routines, and an example of its usage.

Move:
This is a useful graphics routine and is used in conjunction with other routines. A sample line of code to call it is:

CALL Move(RP&, X, Y)

where RP& is the RASTPORT pointer initialized above, X is the horizontal or X-axis position, and Y is the vertical or Y-axis position. This routine simply moves the cursor to the current X, Y pixel location.

After this routine is called, the PRINT statement in AmigaBasic will print from that position. Therefore the PRINT AT (112, 87); "Hi there" statement used in ABasiC would become CALL Move(RP&, 112, 87): PRINT "Hi there" in AmigaBasic. This routine will also affect any AmigaBasic statements with the STEP

argument, as the last point will now be the point specified in the Move statement. Finally, for ABasiC users, the Move statement is the same as ABasiC's LOCATE (X,Y) statement.

SetDrMd:
This routine sets the drawing mode. It affects every operation that outputs to the screen. Thus it affects the AmigaBasic PRINT, LINE, CIRCLE, and other graphics routines. It would be called in this manner:

CALL SetDrMd(RP&, mode)

where once again RP& is the RASTPORT pointer to the current window, and mode is a number composed by adding one or more of the following number together.

0 = JAM1 (just the foreground color is printed onto the background)
1 = JAM2 (both foreground and background colors are printed)
2 = COMPLIMENT (XOR mode or every pixel is complemented)
4 = INVERSE VIDEO (foreground and background colors are exchanged)

Thus the INVERSE(1) function in ABasiC becomes CALL SetDrMd(RP&, 4) in AmigaBasic, and INVERSE(0) becomes CALL SetDrMd(RP&, 0). The INVERSE(1) causes printed text to be printed in inverse video and INVERSE(0) reverts text to normal video for those people who have not used ABasiC.

Another use of SetDrMd is to draw lines, circles, or boxes on a background without disturbing the background. This use is accomplished by setting the drawing mode to COMPLIMINT with CALL SetDrMd(RP&, 2), and drawing every line, circle or box twice. The first time the line is drawn in, and the second time the line is erased leaving the background untouched. Finally, the DRAWMODE command is the ABasiC equivalent of SetDrMd.

Flood:
AmigaBasic already has a PAINT or Floodfill routine, but anyone who has used it realizes that it does not work as you would expect it to work. Fortunately, the graphics ROM routines contain a Floodfill function that works as you expect a Floodfill to work.

Flood fills an area, where all adjacent pixels of the same color
as the one specified at location X, Y are replaced with the colored pattern set by PATTERN, or a plain color if no pattern is set. It is called as follows:

CALL Flood(RP&, 1, X, Y)

Flood can also work like AmigaBasic's PAINT routine if it is called like this:

CALL Flood(RP&, 0, X, Y)

SetOPen:

If you wish to have an area outlined that has been either AREA or Flood filled use the SetOPen routine. SetOPen sets the outline pen color. This outline pen is used to outline an area filled with AREA statements, or an area filled with the Flood routine. It would be called like this:

CALL SetOPen(RP&, color)

where color is the color register number for the outline. This routine is the same as PENO in ABasiC. Also, the PENA and PENB statements in ABasiC are set in AmigaBasic using the COLOR statement, where COLOR A,B would set PENA to A, and PENB would be set to B.

SetSoftStyle:
This routine is not implemented in ABasiC and is a welcome addition to AmigaBasic. SetSoftStyle changes the style of text output to the screen. It affects the PRINT statement and AmigaBasic's output itself. It is called as follows:

```
CALL SetSoftStyle( RP&, style, 255)
```

where style is a number composed of one or more of the following numbers added together.

0 = normal text
1 = underlined text
2 = boldfaced text
4 = italicized text

Therefore CALL SetSoftStyle(RP&, 3, 255) would change the text output to underlined and boldfaced text. With italicized text, if strings are printed out they look correct, but single characters will overlap unless Move is used to put a larger space between each character output.

The last two graphics routines that I will describe MUST be used with the utmost caution. These routines draw lines on the screen in a differant manner than the LINE function in AmigaBasic. But, if a line is drawn outside of the limits of the current window with these routines the system will crash and reboot.

Draw:
This routine draws a line between the last point used in a Move or Draw routine and the point specified by X, Y. It is called in this manner:

```
CALL Draw( RP&, X, Y)
```

It is the AmigaBasic equivalent of DRAW (TO X, Y) in ABasiC.

PolyDraw:
This routine draws a polygon between the points in the two dimensional integer array specified. It is called like this:

```
CALL PolyDraw ( RP&, NumPoints, VARPTR ( array%(0,0) ))
```

It is the AmigaBasic equivalent of MAT DRAW NumPoints, array% in ABasiC.

Well, that is the end of this tutorial. This should answer some of the questions people without ABasiC might have about its foreign graphics commands, and this tutorial should allow people who have ABasiC to use the routines they lost in AmigaBasic. •AC•

Amazing Writers

We are looking for you....

Isn't it great to have the power of an Amiga and then be able to extend your knowledge through trial and implementation? There you are with a brand new way of utilizing the machine we have all grown to love, and you are keeping the information to yourself!

Is this fair?

Is this right?

Of course not!

You have done the work, now take your bows and earn a little **green** for the effort.

Amazing Computing is dedicated to making the Amiga understandable to all, we need your input. We are looking for the "doers" who have completed the small utilities and breakthroughs that were were required to finish their projects. Why not help the rest of us? Think of the prestige in being printed in a magazine that now reaches Amiga owners across the globe!

We do pay our writers! Ok, you will not get rich, but we do try to compensate you for your efforts.

This is your time to show your stuff, not as an experienced writer, but as an Amiga user that knows how to untangle at least one small section of this fascinating machine.

If interested pleases mail a letter to:

Editor
Amazing Computing
PiM Publications Inc.
P.O. Box 869
Fall River, MA 02722

Please include your name, address and phone number.

Remember, our only quide line for writers is BE ACCURATE. We can help you with almost everything else, the experiences and insight must come from you. A submission of hardcopy and electronic text files is most helpful.

A Screen Image Printer (SCRIMPER)

by Perry S. Kivolowitz

The Amiga personal computer supports a device independent printer command named, ``PRD_DUMPRPORT." This command is interesting in that by using it, any graphics displayable on the Amiga's screen can be printed on a printer capable of printing bit maps. As it turns out, most printers already supported by the Amiga include this essential capability.

A while back, I wanted to write a program to exploit this (very sophisticated) feature of the Amiga personal computer. At that time, I had not had any experience with device input/output or with the use of requesters. What resulted was a program called ``scrimper" which stands for``SCReen IMage PrintER."

This issue of Amazing Computing contains the source code to scrimper. This article is intended to serve as a user guide and, moreover, an explanation as to how the program works.

In my discussion of scrimper, I will assume a familiarity with messages and message passing (which if you do not have, you can get by reading last month's Miga-Mania). Also, of course, I will assume some familiarity with the C programming language.

The Scrimper User Manual

On many occasions I have wanted to take a snap shot of the screen and send it to my printer. What I wanted was a small program I could load once and then whenever I wanted to dump a screen I could execute one, then immediately get back to whatever it was that I was doing.

Scrimper does exactly this. Once loaded into memory, it will hang around until its close gadget is selected. I did not want this program to occupy much room either in memory or on the screen, so the window scrimper creates is just large enough to contain its title bar, close gadget and back/front gadgets.

When scrimper is not the active window, it does not take up any c.p.u. time. In fact, even when scrimper is the current window, it is waiting for menu events from the IDCMP (Intuition Direct Communications MessagePort) and not using the c.p.u.

When you depress the menu mouse button (or the RMB, right mousebutton), scrimper presents a menu which lists (in depth order) the screens currently present on your Amiga. Each screen will be represented by a line in the menu.

The contents of the line is usually the screen title (for example: ``Workbench Screen"). However, if the program owning the most recently active window on a screen has set its own screen title using SetWindowTitles, the screen title listed in the menu will be that of the program and not the default screen title (for example: The GfxMem program by Louis A. Mamakos sets the screen title to ``Graphical memory display by Louis A. Mamakos").

Select one of the screens from the menu. Releasing the RMB and then depressing it again will reveal the function menu. Having selected a screen here is where you decide what you want to do with it.

This version of scrimper lets you do one of four things:

(1) ``return to screen selection"
Choose this if you made a mistake in screen selection or no longer care to do anything to the screen you selected.

(2) ``selected screen to front"
By choosing this the selected screen snaps to the front (ie becomes the front most screen in the depth ordering of screens).

If the next mouse button you press is the right one (RMB) the workbench screen will switch back to the front. This is handy for checking on the progress of a program running on its own screen and then quickly switching back to the workbench.

If you were to press the left mouse button (LMB) next, the screen you just brought to the front would become the current screen. In fact, where ever the mouse's pointer happened to be when you clicked the LMB, if there was a window there, it would become the current window.

(3) ``print selected screen"
Upon selecting this item, scrimper will bring the selected screen to the front (for half a second). Then the workbench

screen will again be moved to the front. Scrimper was giving you a chance to make sure that you selected the correct screen.

At this point, scrimper will ask you if that is really the screen you want to print. If you say ``no" at this point, scrimper will let you back out to the screen selection menu.

Scrimper will now try to allocate enough chip memory to make a snap shot of the selected screen. If not enough chip memory is available, scrimper will ask: ``Not enough chip memory for asynchronous print. Continue?"

If you say ``no", scrimper will go back to the screen selection menu. If you say ``yes", scrimper will use the actual memory used by the screen for the print out. This means if you modify the screen while scrimper is printing it, you will get unpredictable results.

If there was enough memory to print or, if there was not but you answered ``yes" to the ``Not enough chip memory" requester, scrimper will ask you: ``Remap Transparent Color To White?"

If you say ``no" the background color of the screen (the blue in a standard workbench screen) will be printed as it is. If you say ``yes" scrimper will automatically change the background color to appear white.

This feature reduces wear and tear on your print head as well as prolongs the life of your printer ribbon.

If, after having selected a screen but before selecting a function, you should deactivate scrimper (by activating another window), scrimper will go back to the screen selection menu.

Error messages which scrimper might produce include:

***``Could Not Allocate Printer Port"**

If this requester appears, the CreatePort call needed to create a message port for talking to the printer failed. The only thing you can do is hit the ``ok" button and scratch your head.

*** ``Could Not Open Printer"**

If this requester appears, scrimper could not gain access to the printer device. This could be for a number of reasons:

* Some other program has the printer device open.
* Some other program opened the printer device but did not close it when it terminated.
* The required files are not present on your system disk. These files include the directory ``devs" which should contain ``printer.device" as well as a subdirectory called ``printers." There should be a file within ``printer" which matches the printer you have specified in preferences.

Scrimper can be started from the workbench by creating a tool icon for it. Remember, programs ran from the workbench cannot contain ``printf" statements.

Description Of Scrimper Code

This version of scrimper is divided into two modules, scrimper.c and printer.c. Let's discuss printer.c first since it is shorter and more simple. In fact, we will save the discussion of scrimper.c for next month. I will provide a lot of detail in my discussion.

My goal, in the discussion, is not only to explain how scrimper does what it does, but to provide an in-depth tutorial example on how to program in C and specifically how to program in C on the Amiga.

You should be able to use the following discussion as a jump off point for your further learning and investigation of Amiga programming.

****

First I define a manifest constant called MAX_CREGS to have the value of 32. This represents the maximum number of color registers the Amiga can make use of (without EXTRA_HALFBRITE_MODE). We will use this number in a few places, so it pays to define the constant.

Next comes two static declarations. One for the printer port and the other for the dumprastport I/O packet. When declaring such things as these, I ask myself if the variables I am declaring are going to be used in other modules. If not, I declare them as statics to make their symbols private to the module in which they are being declared.

If I ever need to import the private symbols into another module, I can always remove the static qualifier. The advantage of doing this is I do not need to worry about declaring another module private variable someplace else and have a naming conflict. Also, good programming style dictates that details of a module's implementation should be kept as hidden as possible from other modules (hidden data types).

In this case, there really is not a down side but, in general, the disadvantage to declaring something static is the variable will then occupy space in the object file on disk where perhaps, if they were not declared as static, the variable would be instantiated by the operating system's process loading facility (and not take up any disk space in the program's executable file).

Next, I declare some external procedures. In Manx C this is very important since the routines I am declaring return pointer results. If I did not declare them as external routines returning pointers, Manx would clip the 32 returned address into a 16 bit integer (and, of course print a warning).

I declare CreatePort as a function returning a pointer to a structure of type MsgPort. We will use this to create a message reply port so we can receive results back from the printer device.

I declare FindTask as a function returning a pointer to a structure of type Task. One of the tricks you can learn from the scrimper code is my use of FindTask in a routine which changes the priority of the printer device handler.

As it turns out, when it is installed, the printer device handler runs at far too high a priority. My experience has shown it runs at a priority of 0, while normal user processes run at a priority of -5. This is definitely a problem since, if the printer is active for long periods, it will effectively block a user program from running. I will talk more on this later.

I next declare AllocRaster to return a thing called a PLANEPTR. This is an Amiga supplied typedef which really means ``pointer to an un-signed byte'' (in include/graphics/gfx.h).

Typedefs are a convenient way of cutting down on the number of characters you have to type. For example, in this case we can say PLANEPTR instead of ``unsigned char *.'' Also, PLANEPTR stands out better than the actual type and makes for more understandable code.

After this, comes a few more declarations including the external window pointer ``w'' which is actually defined in scrimper.c.

Close_mask is what I use to keep track of what resources I have acquired (and must ultimately return). Each time I open or allocate something, I set a bit in close_mask. On exit, I simply run through the bits in close_mask. If the specific bit is on, that means I must return some specific resourse.

Rast_count is the same sort of animal as close_mask. Rast_count keeps track of the number of bit planes (or rasters) I have allocated using AllocRaster. I declare and use a counter (instead of using close_mask) since there could be many planes allocated which must be returned. A counter lends itself to keeping track of multiple like resources more so than bit oriented flags.

The two defines declare which bits in close_mask will have what meaning. If bit 0 is on in close_mask, it means I have to deallocate the message port. If bit 2 is on, I must close the printer device. There is a hole (between bits 0 and 2) because, in an earlier version of scrimper, I was keeping track of something else as well.

This is a good example of why defining a symbol to stand for a numeric constant is a good idea. In a glance I can tell which bits in close_mask are available and of those that are in use I can get an idea for what they are being used.

Next come several IntuiText structure definitions which are passed to Intuition as arguments in AutoRequest calls. IntuiText's are used extensively when programming with Intuition so let's talk a bit about them.

The first two values are unsigned bytes which represent which color registers will be used to render the text. The first value represents the color of the text. If the third value is set to JAM2, the second value represents the color of the background of the text. Remember the first and second values represent color register numbers not colors themselves. This means, of course, that if you are rendering on a two bit plane raster, you can not specify a color register of, say, 21.

Other drawing modes include JAM1 (which is used in scrimper) and XOR. See the Intuition documentation for more details about these. One more tid bit about the first two values of an IntuiText: If they are -1, Intuition will use default values.

The fourth and fifth values in an IntuiText represent horizontal and vertical offsets at which the text will begin. The offsets are based from the top left corner of whatever the IntuiText structure is going into.

For example, the ``no" IntuiText is going inside a boolean gadget. If the fourth value in the IntuiText were 0, the text would begin as high as it could within the surrounding boolean gadget. Similarly if the fifth value were 0 the text would begin as far to the left as possible while still staying inside the boolean gadget.

The sixth value in an IntuiText structure tells Intuition what font is to be used to render the text. Specifically, the sixth value is a pointer to a structure of type TextAttr. In this tutorial I will not go into how to open and manage fonts. Suffice to say that if you supply a NULL pointer, Intuition will substitute the default font (for the screen from which your program is being run).

Finally the seventh value is a pointer to a string which comprises the text you want to print. Why CBM declared this as an unsigned character pointer is beyond me. But since it is declared that way, you have to cast your character pointer into an unsigned character pointer to avoid a warning from most C compilers.

As with most instances of strings in the C programming language, you must terminate the string with a NULL byte. This is done automatically when you enclose characters in double quotes (in C).

By the way, recall we want a pointer to a character, and yet in scrimper I put a bunch of characters inside of double quotes. Where do I work in this pointer stuff?

Well, in C the list of characters surrounded by double quotes is actually an expression. The value of the expression is the address (read that: pointer) of (to) the first character inside the double quotes. Voila! Instant pointer!

The last value in an IntuiText structure a pointer to another IntuiText structure. Usually this value is set to NULL as it is in many of the structures in scrimper. However, if you set this value to the address of another IntuiText structure, the second one will be ``chained" to the first. That is, the second will be drawn whenever the first one is drawn.

The starting location of the second structure will be the final starting position of the first structure after its offsets have been computed. There is one example of this in scrimper. The IntuiText ``no_mem2"is chained to ``no_mem." ``No_mem2" will have its top edge 10 pixels lower than the top edge of ``no_mem" where ever it is that ``no_mem" finally ends up.

Now, into the routine dump_screen.

Dump_screen takes five arguments:

(1) s - a pointer to the screen to be printed.

(2&3) bx and by - offsets into the screen of where to begin printing. In this version of scrimper these two parameters will always be zero. But, if you wanted to print only a small section of the screens you'd pass the section's top edge and left edge in by and bx (beginning y and beginning x).

(4&5) width and height - this is the number of pixels in the x and y directions we want to print. In this version of scrimper width is always the width of the screen and height is always the actual height of the screen.

Notice, the only argument I fully specify is ``s." The last four parameters do not have to be explicitly typed since they are integer parameters. Any thing other than integer parameters should be explicitly typed (in fact, if the parameter is a pointer type, it must be explicitly typed to avoid compiler warnings and possible errors).

Actually, if your goal is really portable code you should not even count on the implicit integer data type. For example, the Aztec C compiler (from Manx) defaults to 16 bit integers. What if I passed a number larger than 16 bits can represent? Obviously, some kind of error!

When it comes to argument passing and to typing in general in the C programming language, take a lesson from the March Hare who said, ``Then you should say what you mean."

Next comes some local variable declarations. I declare req to be a pointer structure of type IODRPReq, the request block for asking the printer device to dump a rastport. The initialization is a little funny in that if you were to say:

```
*req = &request;
```

as a normal C statement, you would be saying ``put the address of request into the structure pointed to by req" which is a big mistake. What we intended is make req point to the beginning of request. As a normal C statement this would correspond to:

```
req = &request;
```

The way to rationalize this is that we have just declared a pointer variable (``some_type *some_pointer_variable") so, if we were to provide an initializer what would we initialize? A pointer.

To review this (ahem) pointer:

```
register struct IODRPReq *req = &request;
```

is the same as saying:

```
register struct IODRPReq *req;

req = &request;
```

and NOT the same as:

```
register struct IODRPReq *req;

*req = &request;
```

You might wonder why I go through the effort of setting up req to point to request. Why not use request straight off. Well, two reasons. First I know that I will be doing a lot of references to the address of request when I start assigning

values to the members of the request structure. So, I would like to keep the address of request someplace handy like a register. Second, req is less to type then request!

In the next line the initialization of drp to point to rp is very important. Drp is a pointer to a RastPort structure which is what must be passed to the printer device to tell it what to print. I know that when I make the call to the printer device I will pass it a pointer to one of exactly two RastPorts. Here I am initializing drp to point to one of them (sort of making an assumption). If I find out later that I assumed wrong, I can fill in the other value later.

In general, initializing a variable to its most likely final value is often done in programs. I do it from time to time (like here) as a holdover from my days as an assembly language programmer where it was definately more efficient (one whole instruction!) to do it this way.

One more bit of trivia. A hold over from my days as a Fortran programmer, notice I declared a miscellaneous integer variable ``i.'' Man, old habits die hard!

First, we initialize close_mask to 0. Recall that close_mask will have a bit set on for each resource I allocate to scrimper and must remember to give back to AmigaDos. It would not be too cool to have some stray bits turned on in close_mask when I went to start deallocating things.

We move the selected screen to the front most position so the user can get a look at it. Notice I just go ahead and use the value of ``s.'' Ordinarily I would check first to see if s was a valid pointer by comparing it to NULL (which is used by convention to specify that a pointer has no actual value). I do not do this check in this case because the routine which calls dump_screen checks s for NULL for me.

But, if I had wanted to perform the check here, I would have said something like:

```
if (!s) return;

or
if (s == NULL) return;
```

NULL, by convention has the same value as ``falsity'' so the first statement is the most efficient way of coding a test of a pointer for equality to NULL.

Next, I call the Dos routine Delay. Delay will pause your program for the specified number of clock ticks (which on the Amiga occurs 50 times a second). Notice, I cast the 30 into a long integer by placing an ``L'' after it. All integer arguments passed to any Amiga library routine are expected to be passed as 32 bit values. Since Aztec C defaults to 16 bit, integers the cast to a long is required for properly working code.

The call to Delay keeps the selected screen in front of all others for about half a second. Then, we put the workbench screen back in front.

Next I put up the requester asking if you really wanted to print the screen I was just showing. I do this by calling AutoRequest which is an Intuition call.

Auto request is a very simple way of formatting and displaying very simple requesters. To put up your own requester requires a lot of code to handle IDCMP events coming back as a result of the user clicking gadgets and such. AutoRequest handles all that for us allowing us the luxury of being able to put up a real live requester with only one C function call.

The requesters that AutoRequest can render are, however, limited to a little text (called the ``body" of the requester) and one or two boolean gadgets. In short, requesters rendered with AutoRequest can ask a question after which the user must respond with exactly one boolean gadget selection chosen from at most two boolean gadgets.

Scrimper uses AutoRequests to render both one and two choice requesters. The two choice requester is useful for asking yes or no questions. By convention the ``yes" or ``true" choice is rendered on the left while the ``no" or ``false" choice is rendered to the right.

One choice requesters are useful for putting up some information for the user to read. AutoRequest will not return (and thus your program will not continue) until the user selects the one boolean gadget available. By convention, one choice requesters are always rendered on the right (where the ``false" or ``no" gadget goes).

The boolean gadgets rendered by AutoRequest will always be RELVERIFY gadgets which means you will need to both depress the LMB and release it while over the gadget in order for the selection to be made.

You provide AutoRequest a pointer to the window you want the requester associated with. Also, you pass three IntuiText structures corresponding to the body text, the text for the ``true" gadget and text for the ``false" gadget. The offsets of the body text are based from the upper left corner of the requester so you have control over where the body text goes.

However, Intuition automatically chooses where to place the boolean gadgets based upon the width and height of the requester which you provide. The offsets of the gadget texts are based on where ever Intuition finally places the gadgets.

If your body text is too large for the width you have specified, Intuition will truncate the text to fit the width. One more thing about the body and gadget texts, they will be rendered in the screen's default font if the TextAttr pointer field in the IntuiText structures are equal to NULL. If they are not, Intuition will try to render the texts in the specified font.

When calling AutoRequest, you can also tell Intuition what other kinds of events you want to consider positive and which you want to consider negative. Usually you will specify these flags (called PositiveFlags and NegativeFlags in the documentation) to be zero meaning you want only the boolean gadgets to constitute a response from the user.

If you specify non-zero flags, you should in fact be specifying one of the defined IDCMP flags. For example, specifying PositiveFlags to be DISKINSERTED means that while the requester is up on the screen and a disk should happen to be inserted, you want this to be considered equivalent to the user having selected the ``true" gadget.

AutoRequest returns either true or false depending upon which gadget (or equivalent event) the user selected.

In scrimper, right after moving the workbench screen back into the front most position, I present a requester asking you if the screen just shown is really the one you want to print. If you decide it was not, you would select the ``false" gadget (represented by the ``no" IntuiText structure). This would cause AutoRequest to return false so in the next statement, I would return to the main routine which would accomplish an abort of the print.

If we really want to print the selected screen, I go ahead and initialize my internal rastport using InitRastPort. InitRastPort, by the way, is how the current font will be associated with a new rastport. The purpose of a rastport is to contain a few items of information as well as to point to the BitMap structure which will actually contain the graphical data shown on the screen.

In fact, right after initializing our internal BitMap structure, we make the internal rastport point to the internal bitmap. We still, at this point are assuming we will have enough memory

to make a snap shot of the selected screen.

In the subsequent ``for" loop, I try to allocate as many bit planes as the selected screen has. Notice the compound initializer in the for loop which initializes both ``i" and ``rast_count." You can read the for loop as follows:

```
i = 0;
rast_count = 0;

while (i < s->BitMap.Depth) {

/* all the other stuff */

i++;
}
```

In other words, all for loops can be expressed as while loops as follows:

```
for (initialization; continuing condition; loop control)
statements
```

is the same as:

```
initialization;
while (continuing condition) {

statements;
loop control;

}
```

In the loop, we try to allocate a raster large enough to one plane of the selected screen. If the AllocRaster fails, notice we immediately call free_raster. Free_raster depends on rast_count to be an accurate reflection of the number of rasters ACTUALLY allocated. That is why we increment rast_count after we are sure the allocation was successful and why we were careful to initialize it to zero before the loop started.

If one of the AllocRaster's fail, we ask the user if they want to print using the actual memory of the selected screen. If so we make the``drp" pointer point the rastport of the selected screen instead of our internal rastport. This way, when we actually make the printer call whether we are actually passing the internal rastport or the selected screen rastport, our printer call looks the same.

If drp still points to the internal rastport when we finally fall through the bottom of the loop, it means we will be using the internal rastport and bit map. If this is the case, we need to initialize the internal bit map as well as actually take the snap shot of the selected screen.

Notice right after the opening brace, I declare some new local variables. An opening brace in C is the start of a statement block. Anytime you start a new statement block, you have the opportunity of declaring some more local variables. The variables declared inside of the statement block are, in fact, REALLY local since they are defined only between the starting brace and the corresponding ending brace.

In this case, I know that the variables src and dest are used only within this statement block. Further, the two variables should be placed in registers. It would be a big waste of registers to allocate two to src and dest when the variables are needed in only the smallest of sections of dump_screen. Declaring them from within the statement block gives me the best of both worlds. I get the variables placed into registers and I do not need to tie down the registers for the whole routine.

The assignment of ``cm" is interesting, it is a structure assignment. The C compiler generates the code to transfer all of the bytes in the source structure to the memory set aside for the target structure. The original C language did not have this feature. Structures had to be assigned one component field at a time or copied by some blind byte mover like memcopy. Some years after the original C compilers became available structure assignment (and enumeration types) were added to the ``standard" language.

I do the structure assignment because most of the fields in ``cm" will ultimately have the same value as the ColorMap from the selected screen. So, why copy the whole structure over and then correct what will be different.

In the ``for" loop I am copying over the selected screen's color mappings. Notice the double check I do to decide when to end the loop. I will end the loop when the loop counter reaches the stored number of colors OR the logical maximum number of color registers a screen can have. This way, even if the stored count is erroneous (too large or negative) I will not over run my preallocated buffer. Checking such as this is known as ``firewalling" and is a good practice in loops, especially a loop which modifies memory.

Next, I put up a requester to see if you want to remamp color register zero to white. Color register zero is normally the background color (although using paint programs like Deluxe Paint, you can change this). For example, if you were to print the King Tut picture that comes with D.P. without remapping color register zero, you would get quite a bit of black which translates into heavy ribbon and printer wear.

Unfortunately, some pictures use the background color as part of the overall picture. Again citing King Tut as an example, if you were to remap color register zero to white, you would get white splotches in non-pleasing locations.

The Delay statement is there to allow Intuition time to redraw the workbench screen after removing the latest requester. If this Delay were to be left out and you were printing the workbench screen, the last requester would always be in the picture.

Finally, we take the snap shot of the selected screen with ClipBlit. ClipBlit requires pointers to a source and destination RastPort. Also, you need to pass an x and y offset within the

source RastPort as well as an x and y offset in the destination RastPort. The width and the height of the region you want to operate upon are also passed.

Finally, the last argument to ClipBlit is a specifier which tells the Amiga's Blitter how to perform the copy. In this case, I want just a straight copy which is represented by hexidecimal C0. In specifying the blit operation, only the upper four bits are used (the ``C" digit). A complete discussion of the way to construct these parameters is given in RKM. Suffice to say the digit actually represents a logic equation specifying how the source and destination are to be mixed.

The last statement before the next closing brace, assigns zero to both the beginning offsets since we are using our own internal RastPort, of course we are starting from the upper left corner.

If we are not using the internal RastPort (because there wasn't enough memory available to allocate one) we make the printer i/o request block point to the selected screen's ColorMap directly instead of the internal colormap (which would not have been initialized).

Now we get to open up the printer's message port as well as the print device. In last month's Miga-Mania we discussed the role of message ports in device i/o. We have to declare the message port before opening the device since one of the arguments passed to the OpenDevice call will be what the CreatePortcall returns.

As with other functions which return pointers, in Manx C you have to explicitly declare the function as returning a pointer lest its return value be truncated to the 16 bit integer.

If for some reason I could not allocate the message port, I put up a single gadget requester informing you of this unfortunate development. Note that the single gadget is passed as the ``false" gadget. Intuition expects this convention will be followed.

I discard the return value of the AutoRequest call since it is not needed, a single valued requester can only return a single value. Also, the gadget is there only to allow you to tell scrimper to go and continue after you have read and pondered the weight of the requester body text.

Similarly we open the printer device itself. Notice that after each valid open of a resource, I ``or" in the corresponding bit into close_mask. The syntax:

variable operator= expression;

is a shorthand for:

variable = variable operator expression;So the statement:

close_mask |= PPORT;

is the same as:

close_mask = close_mask | PPORT;

Next I call alter_printer_priority which I will discuss below.

Now comes the time to format the rest of the i/o request block. I fill in the request block field representing the printer reply port with a pointer to the port I allocated for that purpose. The printer device notifies my process when the command I have requested completes or terminates in error by sending a message back to me through the message port I set up previously and specify to the printer device by including its address in the i/o request block.

I fill in the command which in this case is PRD_DUMPRPORT. I tell the printer device which RastPort to print (recall, previous to this I initialized the request block's pointer to a ColorMap structure.

The modes field will be filled in with whatever the selected screen's mode was. Information kept in this field include things like ``is this screen interlaced? Is it high res? etc."

The x and y offset within the RastPort that the printer device should begin printing at is specified in io_SrcX and io_SrcY. If we are using an internal RastPort then these will both be zero. Otherwise, bx and by will be whatever value they were orginally passed in with. I did it this way (passedbx and by as parameters to dump_screen) because I had wished to make the dump_screen upwardly compatible with printing only sections of a screen (for example, one window).

For the purposes of scrimper, ignore DestCols and DestRows. They will be ignored by the printer device since I will be asking the printer device to compute the largest printable area and expand (or shrink) the picture to fit. Ordinarily these fields would specify how many printer pixels wide and high you wanted the print out to be.

We should all pause to think about how sexy the ability of the Amiga to shrink and expand arbitrary bit images to fit into (arbitrary) arbitrary sizes on (nearly) arbitrary printers. It is really tough, for example, to shrink an image in such a way that it remains true to the original. To top it off, we need only make this one (PRD_DUMPRPORT) call to harness this facility. All persons should pause to reflect this feature while facing Los Gatos (or wouldn't it be West Chester in this case?).

The last initialization of the request block is heavy black magic. Simply, I'm asking the printer device to use all of the available space on the printer BUT make sure you do not change the aspect ratio of the picture.

There are multitudes of other flags which can be set in io_Special. Believe me, the discussion of the how io_Special interelates to DestCols and DestRows (and everything else) is best not duplicated outside the Amiga manuals considering the already long length of this article. To explain all the options in a PRD_DUMPRPORT would take another one hundred lines of text.

Finally, we submit the i/o with DoIO. DoIO is synchronous. That is,the DoIO call will not return until the i/o completes (either in error or properly). DoIO takes care of stripping the message coming back from the printer device (on our message port) so we won't have to worry about it.

All that remains is to close down the printer device and return whatever resources I had allocated. The call close_down_printer in turn calls free_rasters to actually give back rast_count bit planes of original width and height. Close_down_printer then closes the printer device and then deletes the allocated printer port.

Notice close_down_printer could be called at any intermediate point in the execution of scrimper whenever an error might have been found in dump_screen. That is, the routine will return only those resources actually allocated to scrimper.

Since the Amiga is a true multi-tasking machine, processes are given priorities under which they share the most prized resource in the machine, the central processing unit.

As it turns out, the printer device when loaded and running, runs at an all too high priority. Normal user process run at a priority of -5. The printer device comes in at a substantially better priority of 0 (the more positive the priority the better). This means that given the choice of running your processes (like Deluxe Paint if you were running it and scrimper at the same time) and the printer device handler, the device handler would win every time. Your processes would run only when the printer device relinquished the cpu for some reason (which would be often enough for you to see, your processes are still running).

Alter_printer_priorty changes the priority of the printer device handler by modifying its task control block. First we find the task control block by calling FindTask giving the name of the printer device process. If FindTask returns a real live task block pointer (which it should since previous to the call to alter_printer_priority we called OpenDevice on the printer which would load and start running the printer device handler if it had not already been in memory) we set its priority down to something more reasonable.

The calls to Forbid and Permit are needed to make sure that between the call to FindTask and the modification of the priority in the resulting task block, the task does not terminate. Forbid prevents the Amiga from changing from one process to another. Permit allows this (mutli-tasking) to happen again.

One last note. Scrimper will leave the printer device driver resident in memory after scrimper terminates. If you want to explicitly purge memory of the printer support (to make more memory available for other things) you would attempt to allocate a ridiculously large amount of memory (say 24 Mbytes).

The Amiga, while trying to honor the request, will ask all expendable drivers to give up the memory in which they are sitting. For this to work, you have to make the memory request after closing the printer device since while someone has the printer device open it is not expendable.

Since the amount of memory requested is ridiculously large, no memory will be allocated so there isn't any resource to give back after the allocation attempt.

Man, I am typed out! Next month we will pick up the explanation of the mainpart of scrimper, scrimper.c. Scrimper.c contains all the menu handling stuff as well as some interesting C language constructs. So, tune in next month and Happy Scrimping!!!

```
#include <exec/types.h>
#include <exec/memory.h>
#include <intuition/intuition.h>
#include <libraries/dos.h>
#include <libraries/dosextens.h>
#include <workbench/startup.h>

/*
**          SCREEN IMAGE PRINTER (SCRIMPER)
**
**              Copyright 1986 By Perry S. Kivolowitz
**
*/
#define             WWIDTH 640
#define             MAX_SCREENS          20
#define             SIZE_MI  (sizeof(struct MenuItem))
#define             SIZE_IT  (sizeof(struct IntuiText))

extern void  *OpenLibrary();
extern void  *GetMsg();
extern void  *OpenWindow();
extern void  *ItemAddress();

int is_cli = 0;
int which_screen = 0;
char *window_title = "SCReen IMage PrintER V0.6
                          (PSKivolowitz)";

extern char  *calloc();
struct          IntuitionBase *IntuitionBase;
struct               GfxBase      *GfxBase;
struct          Window       *w;
struct          Screen       *s[MAX_SCREENS];
char          *stitles[MAX_SCREENS];
struct          IntuiMessage *message;
int           screen_count = 0;

extern          generic_cleanup();
extern          screen_init() , screen_pick();
extern          func_init() , func_pick();

struct IntuiText print_scrn_it =
        { 0 , 1 , JAM2 , 0 , 0 , NULL ,
      (UBYTE *) "print selected screen" , NULL };

struct IntuiText back_2_scrn_it =
        {0 , 1 , JAM2 , 0 , 0 , NULL ,
      (UBYTE *) "return to screen selection" , NULL };

struct IntuiText scrn_2_frnt_it =
     {0 , 1 , JAM2 , 0 , 0 , NULL ,
      (UBYTE *) "selected screen to front" , NULL };

struct MenuItem back_to_screen_select =
    { NULL , -5 , 22 , 0 , 11 ,
        ITEMTEXT | ITEMENABLED | HIGHCOMP ,
        0 , (APTR) &back_2_scrn_it , NULL , NULL , NULL };

struct MenuItem print_screen =
    { &back_to_screen_select , -5 , 11 , 0 , 11 ,
        ITEMENABLED | ITEMTEXT | HIGHCOMP ,
        0 , (APTR) &print_scrn_it , NULL , NULL , NULL };

struct MenuItem screen_to_front =
    { &print_screen , -5 , 0 , 0 , 11 ,
        ITEMTEXT | ITEMENABLED | HIGHCOMP ,
        0 , (APTR) &scrn_2_frnt_it , NULL , NULL , NULL };

struct Menu function_menu =
    { NULL ,
        10 , 0 , 130 , 10 ,
        MENUENABLED ,
        "Select Function" ,
        &screen_to_front };

struct Menu screen_menu =
    { NULL ,
        10 , 0 , 120 , 10 ,
        MENUENABLED ,
        "Select  Screen" ,
        NULL };

struct jmptbl
    { struct Menu *menu;
        int (*init)();
        int (*cleanup)();
        int (*pick)();
    };

struct jmptbl scrn_jmptbl =
    { &screen_menu ,
        screen_init ,
        generic_cleanup ,
        screen_pick };

struct jmptbl func_jmptbl =
    { &function_menu ,
        func_init ,
        generic_cleanup ,
        func_pick };

struct NewWindow nw =
   { 0,10,WWIDTH,10,-1,-1,
    ACTIVEWINDOW | INACTIVEWINDOW |
                CLOSEWINDOW|MENUPICK ,
    WINDOWDEPTH | WINDOWDRAG | NOCAREREFRESH |
       WINDOWCLOSE ,
     NULL,NULL,NULL ,
       NULL,NULL,0,0,0,0,WBENCHSCREEN };

#define      IBPtr      struct IntuitionBase *

alloc_menu_item(p , i , s)
register struct MenuItem **p;
register char *s;
{
        *p = (struct MenuItem *) calloc(1 , SIZE_MI);
 if (*p) {
        (*p)->NextItem = NULL;
        (*p)->Height  = w->WScreen->Font->ta_YSize;
        (*p)->TopEdge  = i * (*p)->Height;
        (*p)->LeftEdge = -5;
        (*p)->Flags= ITEMTEXT | ITEMENABLE |
```

```
HIGHCOMP;
        (*p)->Command      = NULL;
        (*p)->SubItem  = NULL;
        (*p)->NextSelect  = NULL;
        (*p)->MutualExclude = (long)  ~(1 << i);
        alloc_intuitext(&((*p)->ItemFill) , s);
        if (!(*p)->Itemfill) *p = NULL;
        else (*p)->Width = IntuiTextLength((*p)->ItemFill);
         }
}

alloc_intuitext(p , s)
register struct IntuiText **p;
register char *s;
{
        register char *d = NULL;
        register int length = strlen(s) + 1;

        if (length & 1) length++;
        *p = (struct IntuiText *) calloc(1 , SIZE_IT);
        if (*p) {
              d= calloc(1 , length);
              if (d) {
                    (*p)->FrontPen = 0;
                    (*p)->BackPen  = 1;
                    (*p)->LeftEdge  = 0;
                    (*p)->TopEdge  = 0;
                    (*p)->DrawMode = JAM2;
                    (*p)->ITextFont = NULL;
                    (*p)->NextText = NULL;
                    (*p)->IText = (UBYTE *) d;
                    strcpy(d , s);
                     } else *p = NULL;
               }
}

#define       EVEN(S)  ((S) & 1 ? (S) + 1 : (S))

get_screens()
{
  register int i = 0;

 Forbid();
    s[0]       = w->WScreen;
    stitles[0]       = calloc(1 EVEN(strlen(s[0]->Title)+1));
         screen_count = 1;
         if (stitles[0])
               { strcpy(stitles[0] , s[0]->Title);
                 while (s[i++]->NextScreen)
                       { s[i] = s[i-1]->NextScreen;
           stitles[i] = calloc(1 , EVEN(strlen(s[i]->Title)+1));
                       if (stitles[i]) strcpy(stitles[i] , s[i]->Title);
                       else
                             { Permit();
                               return((int) FALSE);
                             }
                       screen_count++;
                    }
               }
         Permit();
         return((int) TRUE);
}
```

```
build_menu(menu , scrn)
register struct Menu *menu;
register struct Screen *scrn;
{
  register int i;
  register struct MenuItem *p;

   if (get_screens())
    { alloc_menu_item(&menu->FirstItem , 0 , stitles[0]);
        if (!menu->FirstItem) return((int) FALSE);
        p = menu->FirstItem;
        for (i = 1; i < screen_count; i++)
          { alloc_menu_item(&(p->NextItem) , i , stitles[i]);
                if (!p->NextItem) return((int) FALSE);
                p = p->NextItem;
          }
        return((int) TRUE);
    }
  return((int) FALSE);
}

main()
{
 extern void massage_left_edges();

 is_cli = 1;
 IntuitionBase =
     (IBPtr) OpenLibrary("intuition.library" , 0L);
        if (!IntuitionBase)
        {
           #ifdef          PRINTF
   if (is_cli) printf("could not open intuition library\n");
           #endif
              exit(1);
           }
 GfxBase =
  (struct GfxBase *) OpenLibrary("graphics.library" ,0L);
   if (!GfxBase)
     {
           #ifdef          PRINTF
    if (is_cli) printf("could not open graphics library\n");
           #endif
              CloseLibrary(IntuitionBase);
         exit(1);
        }
   if (! (w = (struct Window *) OpenWindow(&nw)))
    {
           #ifdef  PRINTF
       if (is_cli) printf("could not open dummy window\n");
           #endif
              CloseLibrary(GfxBase);
              CloseLibrary(IntuitionBase);
              exit(1);
       }
       SetWindowTitles(w , window_title ,  -1L);
       set_widths(&function_menu);
       massage_left_edges(&function_menu);
     while(message=(struct IntuiMessage *) GetMsg
            (w->UserPort)) ReplyMsg(message);
       while (getmenu(&scrn_jmptbl));
            out:while (message=(struct IntuiMessage *)
              GetMsg(w->UserPort)) ReplyMsg(message);
```



```
        SetWindowTitles(w,NULL,NULL);
        CloseWindow(w);
        CloseLibrary(GfxBase);
        CloseLibrary(Intuition);
exit(0);
}
screen_pick(code)
USHORT code;
{
 which_screen = ITEMNUM(code);
 return(getmenu(&func_jmptbl));
}

freeall()
{
 register int i;
 register struct MenuItem *p = screen_menu.FirstItem;
 register struct IntuiText *t;
 register struct MenuItem *pnext;

for (i=0; i<screen count; i++) if (stitles[i]) free(stitles[i]);
 while (p)
  { if (p->ItemFill
    { t=(struct Intuitext *) p->ItemFill;
      if (t)
       { if (t->IText) free (t->IText);
         free(t);
       }
    }
```

```
    pnext = p->NextItem;
    free(p);
    p=pnext;
   }
 screen_menu.FirstItem = NULL;
 screen_count = 0;
}

void massage_left_edges(menu)
register struct Menu *menu;
{
 register int max_width = 0;
 register struct MenuItem *p = menu->FirstItem;

 if (!p) return;
 while (p)
 { if (p->Width > max_width) max_width = p->Width;
   p=p->NextItem;
 }
 p=menu->FirstItem;
 while (p)
 { ((struct IntuiText *) (p->ItemFill))->LeftEdge =
       (max width - p->Width)/2;
   p->Width = max_width;
   p=p->NextItem;
 }
}
```

```
set_widths(menu)
register struct Menu *menu;
{
 register struct MenuItem *p = menu->FirstItem;

while (p)
 { p->Width = IntuiTextLength(p->ItemFill);
   p=p->NextItem;
 }
}

func_pick(code)
USHORT code;
{
 register struct Screen *scrn;
 extern struct Screen *verify_screen_pointer();
 register ULONG class;

 if (scrn=verify_screen_pointer())
 switch (ITEMNUM(code))
 {
   case 2:return ((int) TRUE);

   case 1: ClearMenuStrip(w);
   dump_screen(scrn,0,0,scrn->Width,scrn->Height);
   return ((int) TRUE);

  case 0: if (scrn( == w->WScreen) break;
  ScreenToFront(scrn);
  Wait(1L<<w->UserPort->mp_SigBit);
  message=(struct IntuiMessage *) GetMsg(w->UserPort);
  if (message)
   { class=message->Class;
     ReplyMsg(message);
 if(class!=INACTIVEWINDOW) ScreenToFront(w>WScreen);
     if (class == CLOSEWINDOW) return((int) FALSE);
   }
 return((int) TRUE);
 }
DisplayBeep(w->WScreen);
return((int) TRUE);
}

struct Screen *verify_screen_pointer()
{
 register struct Screen *hoped_for;
 register struct Screen *p;
 if (! (hoped_for = s[which_screen])) return(NULL);
 p = s[0];
 while (p)
   {if (p == hoped_for) return(hoped_for);
    p=p->NextScreen;
   }
 return(NULL);
}

func_init()
{
 ClearMenuStrip(w);
 SetMenuStrip(w , &function_menu);
```

```
 return((int) TRUE);
}

generic_cleanup()
{
 ClearMenuStrip(w);
 freeall();
}
screen_init()
{
 if (!build_menu(&screen_menu , NULL))
     return((int) FALSE);
 massage_left_edges(&screen_menu);
 SetMenuStrip(w , &screen_menu);
 return((int) TRUE);
}

getmenu(jmptbl)
register struct jmptbl *jmptbl;
{
 register struct MenuItem *mi;
 register ULONG class;
 register USHORT code , code_save;
 register int ret_val;
if (! (*jmptbl->init)()) return((int) FALSE);
         while (1)
{
message = (struct IntuiMessage *) GetMsg(w->UserPort);
         if (!message) {
          Wait(1L << w->UserPort->mp_SigBit);
               continue;
}
   class = message->Class;
   code  = message->Code;
   ReplyMsg(message);
   switch (class)
               {
               case CLOSEWINDOW:
                    (*jmptbl->cleanup)();
                    return((int) FALSE);

               case INACTIVEWINDOW:
               case ACTIVEWINDOW:
                    (*jmptbl->cleanup)();
                    return((int) TRUE);

               case MENUPICK:
                    if (code == MENUNULL) continue;
                    while (code != MENUNULL)
    {
        mi = (struct MenuItem *)
        ItemAddress(jmptbl->menu , (long) code);
                         code_save = code;
                         code = mi->NextSelect;
    }
                    ret_val = (*jmptbl->pick)(code_save);
                    (*jmptbl->cleanup)();
                    return(ret_val);
               }
         }
}
```

```
#include <exec/types.h>
#include <exec/memory.h>
#include <exec/tasks.h>
#include <intuition/intuition.h>
#include <devices/printer.h>
/*
** SCREEN IMAGE PRINTER (Printer.c)
**
**      Copyright 1986 By Perry S. Kivolowitz
**
**The author grants permission to duplicate and reproduce
**this software provided that no commercial gain can be
**had as a consequence of use or reproduction of this
**software and that this and other identifying informat-
**be left intact.
**
**This software is provided as a service to the readers
**of Amazing Computing and does not imply a commitment
**on behalf of the author to provide support nor can the
**author be held liable for the consequences of use or
**misuse of this software.
**
*/

#defineMAX_CREGS32
static  struct IODRPReq  request;
static  struct MsgPort   *pport;
extern  struct MsgPort   *CreatePort();
extern  struct Task      *FindTask();
extern  PLANEPTR          AllocRaster();
extern  struct Window    *w;

static  struct BitMap    bm;
static  struct RastPort  rp;
static  int   close_mask;
static  int   rast_count;

#definePPORT0x0001
#definePRINTER0x0004

static struct IntuiText no_port =
    { 0,1,JAM1,10,16,NULL,(UBYTE *)
   "Could Not Allocate Printer Port" , NULL};

static struct IntuiText no_lp =
    { 0,1,JAM1,20,16,NULL,(UBYTE *)
   "Could Not Open Printer" , NULL};

static struct IntuiText no_mem2 =
    { 0,1,JAM1,10,16,NULL,(UBYTE *)
   "asynchronous print. Continue?",NULL};

static struct IntuiText no_mem =
    { 0,1,JAM1,30,5,NULL,(UBYTE *)
   "Not enough chip memory for",&no_mem2};

static struct IntuiText remap =
    { 0,1,JAM1,17,10,NULL,(UBYTE *)
   "Remap Transparent Color To White?",NULL};
```

```
static struct IntuiText ru_sure =
    { 0,1,JAM1,22,5,NULL,(UBYTE *)
    "Print The Screen?" , NULL};

static struct IntuiText yes =
    { 0 , 1 , JAM1 , 6 , 4 , NULL , (UBYTE *)
    "Yes" , NULL};

static struct IntuiText ok =
    { 0 , 1 , JAM1 , 7 , 4 , NULL , (UBYTE *)
    "OK" , NULL};

static struct IntuiText no =
    { 0 , 1 , JAM1 , 7 , 4, NULL , (UBYTE *)
    "No" , NULL};

dump_screen(s , bx , by , width , height)
struct Screen *s;
{   register struct IODRPReq *req = &request;
    struct RastPort *drp = &rp;
    struct ColorMap cm;
    unsigned short color_table[MAX_CREGS] , *src , *dest;
    int result;
    int i;

    close_mask = 0;
    ScreenToFront(s);
    Delay(30L);
    ScreenToFront(w->WScreen);
    result=
    AutoRequest(w,&ru_sure,&yes,&no,0L,0L,200L,60L);
    if (!result) return;
    InitRastPort(&rp);
    InitBitMap(&bm,(long)s->BitMap.Depth,
            (long)width,(long)height);
    rp.BitMap = &bm;
    for (i = 0 , rast_count = 0; i < s->BitMap.Depth; i++)
    { bm.Planes[i] = AllocRaster((long) width,(long) height);
      if (!bm.Planes[i])
        { free_rasters(width , height);
          result =
           AutoRequest(w,&no_mem,
                       &yes,&no,0L,0L,300L,60L);
          if (!result) return;
          drp = &s->RastPort;
          break;
        }
    rast_count++;
    }
src = (unsigned short *)
    s->ViewPort.ColorMap->ColorTable;
dest = color_table;
cm = *s->ViewPort.ColorMap;
for (i = 0; i < cm.Count && i < MAX_CREGS; i++)
    *dest++ = *src++;
cm.ColorTable = (APTR) color_table;
if (AutoRequest(w , &remap , &yes, &no , 0L , 0L ,
            320L , 60L))
{
    /* make trnsprnt color into white */

    color_table[0] = 0xFFF;
}
 if (drp == &rp)

{ Delay(60L);
 ClipBlit(&s->RastPort,(long)bx,(long)by,drp,0L,0L,
        (long)width,(long)height,0xC0L);
  bx = by = 0;
 }
if ((pport = CreatePort(NULL , 0L)) == NULL)
{
    (void)
    AutoRequest(w,&no_port,NULL,&ok,0L,0L,300L,60L);
    close_down_printer(width , height);
    return;
}
close_mask |= PPORT;
    if (OpenDevice("printer.device",0L,req,0L))
{
    (void)
AutoRequest(w,&no_lp,NULL,&ok,0L,0L,300L,60L);
    close_down_printer(width , height);
    return;
}
close_mask |= PRINTER;
    alter_printer_priority();
    req->io_ColorMap = &cm;
    req->io_Message.mn_ReplyPort = pport;
    req->io_Command = PRD_DUMPRPORT;
    req->io_RastPort = drp;
    req->io_Modes   = s->ViewPort.Modes;
    req->io_SrcX = bx; req->io_SrcY = by;
    req->io_DestCols = req->io_SrcWidth = width;
    req->io_DestRows = req->io_SrcHeight = height;
    req->io_Special = 0x8C;
    DoIO(req);
    close_down_printer(width , height);
}

close_down_printer(w , h)
{
    free_rasters(w , h);
    if (close_mask & PRINTER) CloseDevice(&request);
    if (close_mask & PPORT) DeletePort(pport);
}

free_rasters(width , height)
{
register int i;

    for (i = 0; i < rast_count; i++) FreeRaster
        (bm.Planes[i] , (long) width , (long) height);
}
alter_printer_priority()
{
register struct Task *pd;
    Forbid();
    pd = FindTask("printer.device");
    if (pd && pd->tc_Node.ln_Pri >= 0)
        pd->tc_Node.ln_Pri = -5;
    Permit();
}
```

•AC•

Microsoft Basic Tutorial Part II: MBASIC Commands

by Kelly Kauffman
CIS 70206,640

This time around, you will be familiarized with some of MBasic's do's and don'ts for programming. If there was ever a time to take a look at a MBasic program listing, this is it. Load up any program in your "BasicDemos" drawer and then LIST it like I showed you in last month's installment.

Take special notice of one thing, there are NO line numbers. Line numbers were used (and still are) in some Basic's on other machines (even the old ABasiC with rev. 1.0 of the WB and Kickstart). They tell the computer what order to execute commands in. Things are different now. Line numbers are now completely optional. The program will execute from the top-down, and it will process each command as it runs into it. Now you may be asking yourself, "What if I have to want the program do something, say, 30 or 50 times?" Well, I'm glad you asked.

A SET of commands can be given a name. When you name a group of commands, this section becomes what is called a "Procedure." An example follows:

```
Hello:      (Procedure name followed with a ":")
    PRINT "Hello Amiga User!"
    PRINT "Having fun???"
```

Note that you MUST follow a procedure's name with a colon. Now that is fine and dandy, but what if, like you asked before, you want to do that forever and ever (or at least until we stop the program)?

You would stick in after the second PRINT statement the following:

```
    GOTO Hello
```

So that your entire listing would look like this:

```
Hello:
    PRINT "Hello Amiga User!"
    PRINT "Having fun???"

GOTO Hello
```

Now when the program is RUN, this would print the words in quotes forever, or at least until you press CTRL-C to stop the program. See how simple it is???

It is important to note that if you RUN the program without the GOTO command, the computer will automatically stop and return control back over to you.

Now say you wanted it to print something else besides that. That's simple too. Simply move your cursor up to the word "User!" in the second PRINT statement using the four arrow keys on your keyboard. Press the "BACK SPACE" key until all that's left is "Hello " (Note to still leave the space after the word Hello.) Now type "Amazing Computer Reader!" (Without the quotes.) Your listing should now look like this:

```
Hello:
    PRINT "Hello Amazing Computer Reader!"
    PRINT "Having fun???"

GOTO Hello
```

NOW when you RUN the program, it will print out the new information.

YOUR FIRST COMMANDS:

As you can tell the command PRINT writes whatever follows in quotes on the screen. It is MANDATORY, for now, that you enclose what you want it to write on the screen in quotes.

The GOTO command will make the computer stop at the GOTO command and continue running the program at the procedure name that you specify AFTER the GOTO command.

IT IS VERY IMPORTANT that you do not name your procedures the same as the MBasic commands. For instance, don't name a procedure PRINT: or GOTO: -- this is one of the best ways to confuse the computer and generate an error. These commands (PRINT, GOTO, etc.) are called

RESERVED WORDS. For a complete list of Reserved Words, consult your Amiga BASIC manual, page A-13 and A-14.

One thing that I cannot stress enough is experiment. BUT BEFORE YOU GO GALAVANTING AROUND YOUR KEYBOARD, MAKE ABSOLUTELY, POSITIVELY SURE THAT YOUR DISK(S) IN YOUR DRIVE(S) ARE WRITE-PROTECTED!!!!!!!! In other words, slide the little tabs on the back's of the disks so that you CAN see through the little hole. Now you can play to your hearts content and not worry about losing any information. Besides, it's ALWAYS a good idea to make and use Back-Up copies of your disks.

Now go ahead, LOAD up a program, LIST it, look at all the different commands that they use. Look 'em up in your manual and try to figure out just exactly what they're doing. I'm a firm believer in the experimentation method.

Take statements from the LIST window, and type them into the "BASIC" window and hit return. If they generate an error, big deal. If it does something, make a mental note of it. Then play around with that one for a bit, and move on. This is the best way to begin building your knowledge of AmigaBASIC.

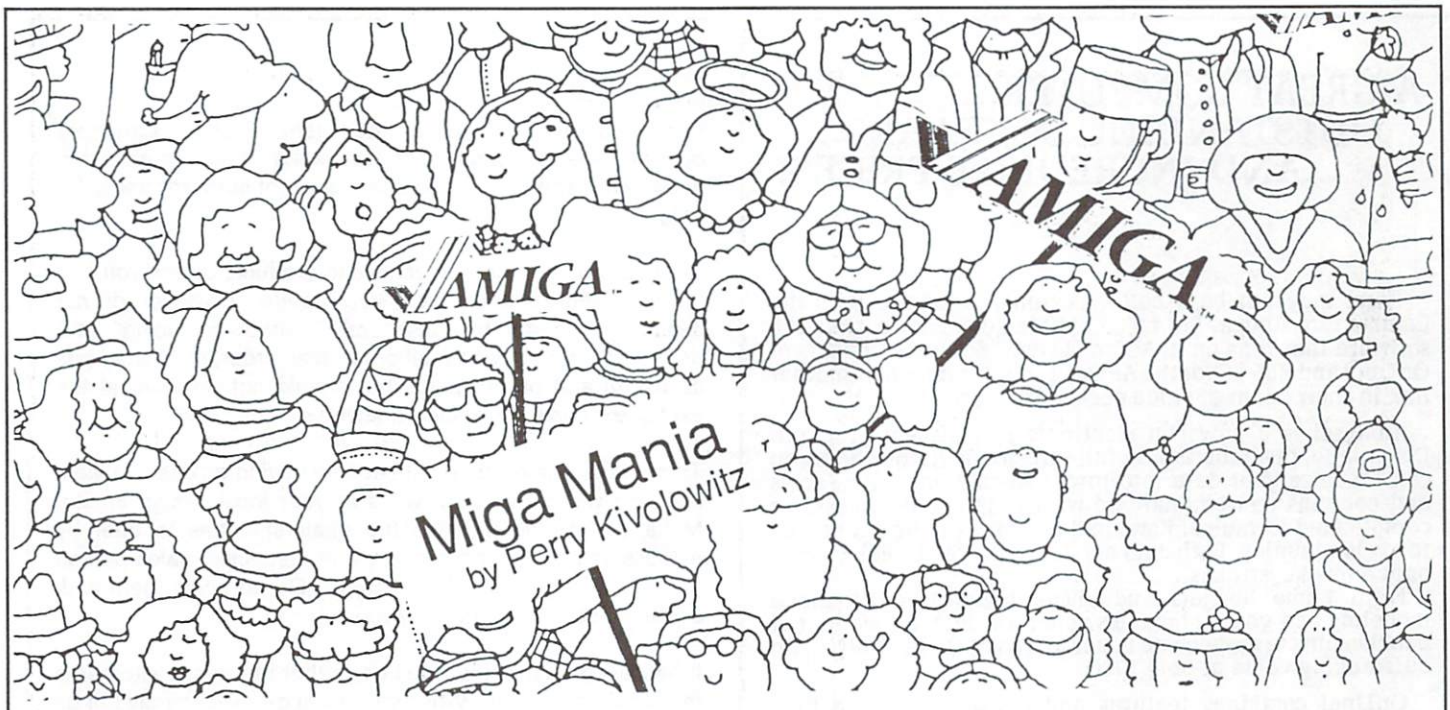
I will be back again next month to explain to you more MBasic commands and tricks!!!

Until next time, here is a quick little program you can type in and play around with.

Guess Number
by Kelly Kauffman
CIS 70206,640

```
RANDOMIZE TIMER

setup:

    CLS
    PRINT "Guess the Number between 1 and 100"
    a$="Guess the Number between 1 and 1 hundred."
    x$=TRANSLATE$(a$)
    SAY x$
    x=INT(RND*100)
    gs=0

guess:

    PRINT "What is your guess ";
    INPUT g
    gs=gs+1
    IF g>x THEN
        PRINT "Too High!"
    a$="Too High."
    x$=TRANSLATE$(a$)
    SAY x$
    END IF
    IF g<x THEN
        PRINT "Too Low!"
    a$="Too Low."
    x$=TRANSLATE$(a$)
    SAY x$
    END IF
    IF g=x THEN
        PRINT "You got it in";gs;"guesses!"
        PRINT "Congratulations!"
        a$="You did it!"
        x$=TRANSLATE$(a$)
        SAY x$
        PRINT
        PRINT "Want to play again (Y/N)";
        INPUT y$
        y$=UCASE$(y$)
        IF y$="Y" THEN GOTO setup
        IF y$<>"Y" THEN END
        END IF

GOTO guess
```

•AC•



Currently, most people purchase their Amiga's from computer specialty stores. Competition against such stores from either other local computer specialty stores new to the Amiga scene or from mail order houses will undoubtedly become fierce in the time to come (if it hasn't already!).

In the long run competition between vendors who want our money can only help, us,the consuming public. But, is bottom line sticker price the only criteria we should use in choosing who to buy from? Is there room in the cut-throat world of computer retail for ``dealer loyalty?"

If you are dealing with a computer store you are doing so out of a sense of dealer loyalty even if you do not recognize it. Why buy the Delux Fence Maker from your dealer if you can get five bucks off from the ad in the back of Zorkmid Magazine? ``Well", you say to yourself, ``if I have a problem with my Fence Maker I can go to ol'Bopka who'll give me my hard earned fazoolas back or will exchange my copy of the Fence Maker for another one on the spot".

This is an example of dealer loyalty. Knowingly paying a little extra for a product to ensure that someone acting in good faith will stand behind the product you purchased.

The basis of dealer loyalty is the interhuman loyalty you build between yourself and another individual. Better dealers often encourage their sales people to deal with specific customers on a lasting one-to-one basis.

The resulting relationship is profitable for both consumer and vendor since you get a sales person who understands your needs and the dealer gets faster and more reliable sales for exactly the same reason.

By purchasing computer products through a flesh and blood dealer, you have the opportunity to asses the worth of a product before you buy. Also, before you spend your money, you have access to a local expert, your steady sales person. After you purchase a product you have a recourse against faulty or less than useful products in the form of the dealership at which you made the purchase.

To help your local dealership work for you, there are several things you can do:

(1) If you do not already deal with one specific sales person, try to establish such a one-on-one liason. Dealing with the same person on a regular basis is the key to a dealership's ability to match your needs to products.

(2) Communicate your needs fully to your steady sales person. Take the time to explain to them your needs, what your expectations are, and how you see your Amiga personal computer fitting in.

(3) If you become aware of products you are interested in that your local dealership does not have, don't keep it a secret. Consumers often get a headstart on new products over sales organizations. Do not assume a policy decision has been made somewhere to not carry a product when what is more likely the case, the dealership just is not aware of the new product.

(4) Provide your sales person with feed back. Were you happy with the product you were sold? An answer of yes or no is useful information to your sales person. Someone else's bad reaction to a product may influence your sales person to not suggest that product for your use. And, your glowing report of another product may help steer others to the same product.

Local computer specialty dealerships offer significant opportunities to consumers to make better purchasing decisions (especially in the aftermarket of software and add-ons).

Take advantage of ready access to products before you buy to ensure the products meet your needs. However, do not misuse your dealer by consistently ``window shopping" at a dealer and then buying mail order. Mail order, of course, has its useful and proper place but should not be used at the explicit expense of your local dealer.

To sum up, one of the most capable buying guides available is found within the shop walls of your local Amiga dealer. Make use of the facilities the dealership has to offer by establishing a personal rapport with a specific sales person (or small group of sales people). Get to know them and, certainly, let them get to know you.

It has been said in the retail trade that the best customer is the repeat customer. While your local dealership may not be superior to mail order on the basis of price, the inter-human qualities of service and support are often far more valuable. It is upon what transpires between the sales person and you that the dealership rests its efforts to make you a repeat customer. Avail yourself!

Due to the size of my Scrimper article for this month's Amazing Computing™ and the shortness of life, this month's Miga-Mania will be short!

User Group News #3
Champaign County Amiga User's Group
The CCAUG meets the first Tuesday of each month at 7 p.m. in Urbana Illinois.
Contact:

Wayne Hamilton
907 West Nevada
Urbana Il, 61801
or
Russ Jacobson
402 McArthur Drive
Urbana, IL 61801

Wayne can be reached during the day at (217) 333 8703 while Russ can be reached (also during the day) at (217) 344 1481. These numbers are where these guys work so be nice!

Keep those cards and letters comin'! See you next month with more Miga-Mania!

•AC•

Over 800 Songs

SYMPHONY MUSIC LIBRARY

Over 30 Hours of Music

The library contains over 800 four voice stereo music pieces to turn your AMIGA into a sophisticated Jukebox providing you with over 30 hours of music.

* Turn your AMIGA into an incredible music machine.
* Over 800 masterpieces in entire library.
* Over 100 songs in each volume.
* Not just single note music, but 4 full voice arrangements.
* Each volume contains over 3 hours of music.
* Stereo, Mono, and MIDI output simultaneously.
* You may specify sound of each of the voices.
* Over 30 hours of music in entire library.
* User selectable tempo.
* Selections may be played thru any MIDI synthesizer using the MIDI CONNECTION (sold separately).
* User selectable key (transposition).
* Includes Jukebox program to allow you to:
  1) Select all pieces and sit back to hours of uninterrupted music.
  2) Select the order you wish the pieces to be played.
  3) Select the number of times you wish each piece to be played.

SYMPHONY LIBRARY Vol 1
Music from Stage, Screen & TV
Pop Music from the 50's, 60's, & 70's
Old Time Favorites
Classical
Christmas (popular & traditional)
Patriotic
Polka Party

SYMPHONY LIBRARY Vol 2
Music Potpourri
Barbershop Quartette
Religious Music
Hits of the 80's
Variety

SYMPHONY LIBRARY Vol 3
Pop Music from the 40's, 50's, 60's, 70's
Beatles Medley

SYMPHONY LIBRARY Vol 4
TV Theme Music
Beethoven, Broadway, & Blues
Kenny Rodgers Greatest Hits
Best of the Beatles
Country Classics

SYMPHONY LIBRARY Vol 5
Popular Potpourri
Polka Party
Classical
Tribute to Bach
Christmas Treasure

SYMPHONY LIBRARY Vol 6
Nostalgic
Richard Rogers Songbook
Music From the Movies
Pop Music from the 60's, 70's, & 80's

SYMPHONY LIBRARY Vol 7
Variety
Classical
Pop Music from the 30's thru 80's
Popular Classics

SYMPHONY LIBRARY Vol 8
Surprise!

Each Volume ~~$39.95~~ $29.95 Introductory Offer

THE MIDI CONNECTION

Attention MIDI Users!

The MIDI CONNECTION is a hardware interface to allow you to connect the AMIGA to any MIDI synthesizer, drum machine, sampler and more. MIDI IN and OUT connectors with cables allow easy installation. The MIDI CONNECTION may be used with our SYMPHONY MUSIC LIBRARY or the MIDI SYMPHONY.

~~$49.95~~ $39.95 Introductory Offer

THE MIDI SYMPHONY

The MIDI SYMPHONY program and the MIDI CONNECTION hardware interface give you the features you need to compose serious music using MIDI equipment. Features include:

- Supports up to 16 tracks.
- 10,000 events per track.
- 40,000 events all tracks.
- May be used as a sequencer.
- Menu driven.
- Overdubbing.
- User friendly graphics display.
- Metronome available for synchronization.
- Tempo may be modified.
- Change key (Transposition).
- Quantizing to 32nd or 64th.
- Playback any or all tracks at any tempo.
- Tracks may be deleted, copied, transposed or mixed.
- Filter out unwanted channel or type of MIDI data.
- Simple music editing.
- Real Time recording and playback.
- Playback all or individual tracks.
- Track delete, copy, transpose or mix.
- Editing

~~$99.95~~ $89.95 Introductory Offer

COMING SOON

SYMPHONY SOUND SAMPLER. This high speech digital sampler allows you to create and modify sounds. When used with the SYMPHONY Piano Keyboard, you can turn your AMIGA into an Ensoniq Mirage.

SYMPHONY WRITER. A composition program specifically for developing sheet music.

SYMPHONY 61 note Piano Keyboard. This professional quality keyboard transforms your AMIGA into a real synthesizer.

CASIO CONNECTION. This program and the 61 note keyboard allows CZ-101 and CZ-1000 owners to use a full size keyboard.

We accept CASH, CHECK, COD, VISA and MASTER CARD orders.
Shipping and handling US and Canada $3.00
Shipping and handling outside the US and Canada $5.00
COD charge $2.00
Illinois residents add 6¼% sales tax.

VISA MasterCard

Dealer Inquiries Invited

Speech Systems
38W255 DEERPATH ROAD
BATAVIA, ILLINOIS 60510
(312) 879-6880

The AMICUS NETWORK

by John Foust

There is much good news for the AMICUS Network and Amazing Computing magazine. First, the public domain software library has grown.

Also, Amazing Computing is now online on the People Link network. If you have a modem for your Amiga, you can join many other Amiga fans in the two Amiga sections of the Commodore Club.

The current AMICUS software library is available for downloading in Section 15, plus the software on the Fred Fish disks. As new software arrives, it will be added to the library immediately.

If you want just one AMICUS program, there would be no need to order the AMICUS disk, or find it at a user group, since you could get a copy in a few minutes on People Link.

The best thing about People Link is the price. Online charges, including packet network costs, are currently $4.25 an hour in the evenings, for any baud rate, 300, 1200, or 2400 baud. Compare this to other networks, which might charge as much as $15 an hour in the evenings.

I have also heard a rumor that People Link is planning to allow uploading of software, to the People Link computers, without charging for the time it takes. So, if you have new public domain software, it would not cost extra to share it with other People Link members. By the time you read this, this change may be in effect. Then again, it might be just a rumor...

People Link sign-up

You can get a People Link account for no charge, and sign-up includes a free hour of online time, to help you get acquainted with the system.

Sign-up is automated. Call 1 (800) 826-8855 with your modem, at any baud rate. If you are in Illinois, call (312) 822-9712. The sign-up will only take a few minutes. It entails giving your name, address, phone and preferred billing method. A credit card is best, since your charges will be posted automatically. Or, you can pre-pay your account, by check. If you would just like a free demo account, you can ask for that, too.

In a few days, you should receive a preliminary account number and password. You can use this to log in for the first time. You should get an information packet with instructions for the system, and a list of local access numbers to People Link. In most cities, you can reach People Link with a local call to Tymenet or Telenet.

On log in, you can type 'GO 68' to get to one of the Amiga clubs. 'GO 68' brings you to section 14, the original People Link Amiga section. 'GO 680' (note the extra zero) will go to the Amazing Computing section. You can type HELP at any prompt, for more assistance.

Public Networks

Public electronic networks, such as People Link, CompuServe, the Source and Delphi, are a world unto themselves.

They vary in quality and services. For example, CompuServe offers airline reservations and an online Grolier encyclopedia, and stock quotes. Most offer two forms of electronic messaging. One is private, user-to-user mail messaging, and another is messaging in a public context.

The public messages are usually grouped into forums or clubs. People Link has a Commodore Club, which has two sections for the Amiga. CompuServe has the

AMIGAFORUM, and Commodore itself had a forum, plus the Commodore discussion maintained in the Toronto Pet User Group, TPUG.

In a club or forum, you can read the messages from hundreds of network members. Some people never speak up, and add to the discussion, and that's OK. Many people are overwhelmed by the complexity of some of the ongoing discussions. Others, such as developers, find this sort of discussion an excellent way to answer questions quickly.

Once, I needed some information on Amiga Basic, long before it arrived in the version 1.1 upgrade. I needed to know the syntax of the sound commands, for an article that was due.

I posted a message early one morning, and by mid-morning, I got a phone call from someone who had an early copy of the manual. A few minutes later, I got a private mail message, with the answers to my questions.

Several months later, I was able to return the favor. This person - another writer himself - needed details of using sprites in C. I had a tutorial on my computer, and I sent a copy of it to him via electronic mail.

Online friends

One of the more popular attractions on People Link is the PARTYLINE. Similar to CB on CompuServe, PARTYLINE allows People Link users to gather and talk to each other in real time.

Everything you type is echoed to every other user in a line, and vice-versa. Each text is prefaced by the "handle" of the speaker, so you can follow the conversation. You handle defaults to your login name, but it can be changed. There are 99 lines available on People Link.

PARTYLINE is a stunning experience at first - a dozen people might be holding a conversation at once. Sometimes, it is orderly, but other times, there might be several threads of conversation happening at once. Just introduce yourself, and ask questions. The people there will be happy to help out a novice user.

Some lines are dedicated to certain themes. For instance, line 29 pretends to be a bar - when you listen to line 29, don't be surprised to see people ordering Pan Galactic Gargle Blasters, and asking to pass the popcorn. Say "hi" to the innkeeper, who goes by the handle ARIA WIND, and tell her AFTER TOUCH sent you.

Public messages

Almost all networks allow public and private storage of files, within the computers of the network itself. While private file storage is available, it has little benefit to most people.

Public storage of files and documents is more useful. Usually, each club or forum has its own storage areas.

When a member uploads a new program from their own computer to the network computer, and the file is approved by the people who oversee the club, the new program will be visible to all members.

In this way, new public domain programs are distributed. New programs are quickly cross-posted between the many different networks, and filter up from smaller, local bulletin boards.

Disclaimer

I spend a lot of time on electronic networks, gathering the latest software, and reading and posting public messages to other Amiga users. I subscribe to Compuserve, Delphi and People Link. I cancelled my subscription to the Source, as there wasn't much of interest to me there. But this unabashed plug for People Link is based on two factors. One, it is a nice network, with a good Amiga section. Two, it is cheaper than any other. (Well, three reasons: I'm fond of the place. I have met lots of nice people there.)

New AMICUS programs:

Disk #5

Disk #5 was not full last month. Since then, the latest IFF description texts and example source code have been added. I got these files days before AIN closed temporarily. (At press time, the developer's online information service, the Amiga Information Network, had just closed. Negotiations were pending for a switch to CompuServe.)

Disk #5 also contains the files necessary to write your own printer drivers.

Disk #6

Disk #6 now contains over two dozen IFF pictures from Deluxe Paint and video digitizers. This disk makes a good demo of the Amiga's video capabilities. Of course, there is a lot of fun in cutting, pasting and re-arranging the images with Deluxe Paint - especially the digitized images.

Disk #7

Disk #7 is the Digi-View video digitizer demo disk. This has the widely-circulated demo of HAM pictures, plus several IFF format pictures.

Rick Wirch has been busy making the AMICUS disks more user-friendly, and accessable from the Workbench, at the click of an icon. All text files, including most C source code, are clickable. This brings up the 'more' program to display the file. IconExec, described below, has made many programs available at the Workbench level.

Disk #8
Disk #8 is full of new programs of interest to Amiga Basic users. Amazing Computing author Rick Wirch has converted most of the ABasic programs from AMICUS disk #1 and #2 to Amiga Basic, along with his own contribution from last month's issue, Deluxe Draw. Deluxe Draw demonstrates the power and flexibility of BASIC programs that use these calls.

Over a dozen .BMAP files are provided on this disk. These are needed as an interface between Amiga Basic and hundreds of operating system calls. See Rick Wirch's article on some elementary graphics calls that use this method, or the source code to Deluxe Draw, from last month's issue.

Other new Amiga Basic programs include a rat-maze program from an issue of Kilobaud, circa 1981. It's amazing how long BASIC programs can kick around. I am still waiting for the real moldy, useless games like CIVIL WAR to get ported to the Amiga. Those have been ported to almost every BASIC-toting computer since the PDP-8.

RatMaze is rather unique, for its use of 3-D graphics. It was originally on an Apple. I still have the issue of Kilobaud that carried the source code...

It has a program called YoYo, which pretends to be a zero-gravity yo-yo. It tracks your mouse movements, and the little shaded yo-yo goes winging around, as if it was on a rubber band.

Another new Amiga Basic program is a version of Clue, like the board game of the same name. This game is menu-driven, a twist on the interface of most games.

The executable programs on disk #8 include StarTerm, the first shareware program in the AMICUS library. It is a fully loaded terminal program, including a dialing directory.

Shareware is a type of public domain software. Usually, the shareware software carries a notice asking you to contribute to the authors, if you feel the program is useful to you. Sometimes, the authors are nastier to commercial or governmental users of the software, and demand a payment, and threaten to haunt your conscience for the rest of your life if you don't pay.

I came across a shareware program that asked you to contribute to the American Cancer Society, instead of the authors. Shareware is a twist on software marketing. Marketing a program is a large investment, even compared to the programming time and effort.

Of course, some shareware programs are not commercial quality. The authors prefer small compensation to no compensation - usually, popular public domain programs only bring popularity to their authors.

Browse is a text-viewing program from Louis Mamakos, the author of GfxMem, the graphic memory usage indicator.

Browse starts with a menu bar with "DF0:" and "DF1:" as items. If you select a drive, it adds a new menu bar, to the left of the first, with the files and subdirectories of that drive listed as menu items. This continues until you select a file. Browse displays the file, a page at a time, with the option of paging forward or back. Source code is included.

SaveILBM is a program to turn any Amiga screen into an IFF format file. This is handy for capturing appealing demo screens, for example. Currently, only the executable version is here, but the source should follow soon.

Scrimper is Perry Kivolowitz's screen printer, featured in the last issue of Amazing Computing. Source code is included with the executable.

ScreenDump and PDScreenDump are two programs that will print the current display to the printer. ScreenDump presents a menu bar, with a Print button, and a button to display information about the authors.

PDScreenDump was written by Carolyn Scheppner, of Commodore's tech support group. It allows you to select all the parameters of the screen dump, such as scaling and shading. Scheppner hopes someone will "Intuitionize" the program, since it is presently guided by straight, C input.

KwikCopy is a straightforward disk copy program, that will help rescue failed disks. It copies everything it can, and simply skips blocks that have read errors.

Mike Meyer wrote a Workbench clock that is only as tall as a menu bar, and as wide as five or six characters. Every fifteen seconds, it moves to the front window, and displays the current time. Source code is included.

SetAlternate, SetWindow, and IconExec are from John Toebes. SetAlternate exercises a property of icons to have an alternate image when single-clicked. It takes two icons created with IconEd, and merges them into one.

For example, you can design two trash can icons, open with a closed lid, and one with the lid open. SetAlternate will merge the two icons together. Then, when the closed can icon is clicked once, the open can icon will appear. Clicking twice invokes the program or opens the window, as before.

SetWindow allows programs that do not have icons to run from the Workbench, with a clickable icon.

As Toebes explains in the source code, some part of the operating system is failing to set the ToolWindow field in the startup message that gets passed to the starting program. This program works around that, and sets up a window to your specifications.

IconExec is used in conjunction with SetWindow. It allows EXECUTE command sequences to be started from the Workbench. You fill in the Tool Types fields you see in the Workbench Info menu item. When the icon is double-clicked, the elements in the ToolTypes array are EXECUTEd one at a time.

The 'libdir' program will list the hunks in an object file. Source code is not available at this time, but there are several other programs in the AMICUS library that give examples of disassembling object and load files.

The 'disksalv', by Commodore employee Dave Haynie, is a program to help restore disks with bad sectors. It needs two disk drives, though. It tries to read every sector from a disk, and copies them to another, but does not use the bitmaps from the damaged disk. It rebuilds the bitmaps, and writes that to the restored disk. Haynie has promised a bigger and better version in the future.

'Arc' is a widely used file compression program, like 'sq' and 'usq'. By analyzing a file, it can substitute common sequences of bits with a smaller, unique sequence of bits, and compress the file. For text files, 'arc' can achieve 30-40 percent compression, and somewhat less on binary files. 'Arc' can also archive a number of files together in a single file. This file is easier to transmit across modem lines, since the files within can be de-compressed to their original sizes, removing the constant hassle of chopping files transmitted with the Xmodem protocol.

'Arc' is constantly tweaked and updated by its authors. It is a standard archival method in the IBM PC world, and the Amiga version is compatible with the IBM version.

The texts on Disk #8 include directions for making your own 5 1/4 disk drive, for use with the IBM emulator. See the current Amazing Computing article for a description of this process in more detail. Some incentive for rolling your own drive has been removed by Commodore's reduction in the price of the 5 1/4 drive, of course.

It also includes an explanation of the Guru Meditation numbers, a list of the bugs in Lattice V3.03, methods of allowing quotes in command line arguments, a few preliminary reviews of the MicroForge hard disk drive, and a simple printer-spooler: an EXECUTE file for sending texts to the printer, while you are doing something else.

Disk #9

Disk #9 was not full at press time. It has Wombat Terminal, yet another communications program.

'Image', by Randy Manchester, is a program to turn a Deluxe Paint brush file into C source data statements. In the absence of programmer-level art tools for designing gadgets, 'Image' bridges the gap between C and Deluxe Paint.

Next issue

In the next issue, the COMDEX Spring computer show will be featured here. At last word, Commodore has a display, and planned to fill it with Amiga developers showing their wares. Don Hicks and I should gather a full report of the show.

•AC•

Microsoft CD ROM Conference

by Jim O'Keane

In the first week of March, Microsoft Corporation hosted the first international CD ROM conference. This event was widely publicized in both optical media journals and microcomputer magazines.

Microsoft tried to attract industry leaders to this event. Some people called it the "Woodstock" of the computer industry. In fact, they advertised the conference with a full-page, color ad in Byte magazine. They succeeded to a degree that no one expected, even Microsoft.

Microsoft was expecting five- to six-hundred attendees, and limited the attendance to 800. To their surprise, more than 1000 people turned out to learn if all the press about CD ROMs was true. Sessions for both novices and experts were offered, as well as an exhibition hall with about twenty exhibitors.

The sessions discussed important issues such as a standard file structure for CD ROMs and the use of CD ROM data bases. One of the more interesting applications was a portable aircraft navigation computer, with maps on CD ROM.

In the exhibition hall, conference attendees could try out a sample CD ROM that was given to each attendee. Microsoft also gave each attendee a large textbook on CD ROMs.

The sample CD ROM contained selections from a multimedia encyclopedia. Each page could be displayed on the screen. Certain words were highlighted in the text. If selected with a pointer, these portions of the encyclopedia entry would bring up a related digitized picture. Some entries were animated, or had CD audio to accompany them. It was a very effective demonstration of the possibilities of CD ROMs.

Unfortunately, the demo disc requires an IBM AT, with a Hitachi CD ROM drive, an EGA graphics board, Microsoft Windows, and an optional AT&T Targa display board! For those of you unfamiliar with the above hardware, it would cost about $10,000 for the system described.

It's too bad they couldn't have used an Amiga, but at the time of development there was no way to control an available CD ROM drive with an Amiga.

Microsoft has no plans to release a multimedia encyclopedia, and did not announce any other products using a CD ROM. Many other CD ROM products were demonstrated, including the Grolier CD ROM encyclopedia by Knowledge Set, formerly known as Activenture. Also available was a database from Datext, and a CD ROM drive from RTI.

One available CD ROM holds all the programs in the PC-SIG, a national IBM user group. The PC-SIG disc has about 300 megabytes of public domain programs for the IBM PC.

Also shown were several SCSI compatible drives by Philips, Sony, Denon, and Sanyo. Most were the same size as full-height 5 1/4" drives. Several manufacturers announced a half-height drive for the IBM PC. It is expected to be available in the fourth quarter of this year.

Videotools exhibited their CD Publisher system which allows a would-be CD ROM publisher to use an IBM PC

based system to simulate a CD ROM, and guide the CD mastering process. This system would greatly reduce time and cost involved in preparing a 550 megabyte database.

By far, the most exciting announcements were made by Philips International, one of the major innovators in the optical disc industry. Philips is one of the largest electronics manufacturers in Europe. They produced the first CD ROMs and CD ROM drives.

Philips was largely responsible for the initial CD audio standard and the subsequent CD ROM standard. As followers of the optical disc industry had known for several months, Philips had been working on a new standard for storing information on CD ROMs.

Much to everyone's relief, they finally formally announced CD-I. CD-I an acronymn for "CD interactive." This is the new standard for many types of media stored on a CD disc. It supports many types of data, including audio, video, and program code.

This chart describes the data formats defined by CD-I. (The audio information listed includes the amount of sound a single CD-I disk can hold, for each type).

Audio

Digital	(CD audio quality stereo)	1 Hour
HiFi	(FM radio quality stereo)	2 hours
MidFi	(AM radio quality stereo)	4 hours
Speech	(Phone quality mono)	16 hours

Video

Digitized video

LoRes	384x280 pixels	RGB 8 bits or 15 bits
HiRes	768x560 pixels	RGB 8 bits or 15 bits

Graphics Animation 4,7 or 8 bits

Text Coding

Bitmap	120 meg Chars
System Font	600 meg Chars

Executable Code, within a CD-I player

A CD-I player should be based on the 68000, and capable of understanding the system calls from the OS-9 operating system

Well, if that looks like a mind-full, it is. During the conference, people thought of more and more uses for a standard like this.

Although the graphics resolutions look odd, keep in mind that the height and width include overscan, the

border around your Amiga screen, and that European television has more lines per picture. Most of Europe uses the PAL system, while we use NTSC video definition.

Please note the distinction between a CD ROM drive, and a CD-I player. Many computer types will be able to access information on a CD ROM, using the CD-I standard. A CD-I player is a consumer device, a stand-alone computer, probably without a keyboard as we know it.

The standard for CD-I players requires a 68000 based system, which can understand the operating system calls for the OS-9 operating system. OS-9 is an operating system widely used on Tandy Color computers, and other 680xx family processors.

If you are wondering where this leaves Amiga users, it can be said in three words - "in the pink!" The Amiga is just what the doctor ordered for developing and using CD-I discs. It has a 68000 processor, digital sound output, and hi-res graphics.

With a version of OS-9 on a KickStart disk, it could be made into a CD-I player. It is rumored that the authors of OS-9 are making an Amiga version of their operating system.

Of course, CD-I represents an incredible source of graphics and audio data. Can you imagine an educational game built around a CD-I or CD ROM disc? Don't get excited too soon, since CD-I players aren't expected to be in the stores until the first quarter of 1987, and titles will probably drag behind by another few months.

Meanwhile, if you get an SCSI port for your Amiga, it will only be a matter of time before a CD ROM drive will be available. Besides, you can always use the SCSI port to connect one of those archaic, magnetic hard disk drives with a measly 40 MBytes of storage.

About the author: Jim O'Keane is an engineer for 3M's Optical Recording Project, at the Menominee, Wisconsin plant. He currently is in charge of mastering both CD ROM and CD audio discs. He is a graduate of University of Wisconsin with a BSEE degree, with an emphasis in computer science. Jim has worked with projects ranging from lasography to music software. Readers with an interest in optical media can reach him at:

Jim O'Keane
2421 Fryklund Dr. #10
Menomonie, WI 54751

•AC•

CD ROM :
The New Papyrus?

For those of you unfamiliar with the technology of CD ROMs, here is a short description of CD ROMs, and how they are made.

CD ROMs are an offshoot of compact audio discs, which are in turn an offshoot of the videodisc. Videodiscs were first developed in the late 70s. They hold up to 60 minutes of high quality video, per side. They are noted for high quality audio, and are popular in rock and classical music markets. There are three major US video disc factories, as well as some specialty houses.

CD audio discs use the same basic technology as the videodiscs. Information is represented as a spiral track of pits on a reflective surface. The pits are detected with a laser beam, and converted back into information.

While the videodisc contains an FM modulated video signal, the CD disc contains digitally encoded audio. Audio CD players and discs were introduced in 1982. Up to one hour of music can be put on a CD disc, which only has one usable side.

CD discs also have built-in error correction and detection, which can correct error bursts up to 3500 bits long. About eight thousand titles are available for audio CD's now, and pressing plants cannot keep up with the demand for discs.

CD ROMs are an adaptation of CD audio discs. They have an additional layer of error correction, which gives an error rate of one bit in 10^{13} bits. Up to 550 megabytes of data can be stored on a 12 centimeter disc. Drives were introduced in 1984, and became commercially available in 1985.

The CD ROM production process is long, and fraught with pitfalls. The data is first compiled by the publisher, and organized into an easily accessable form. This usually involves making an index. Later, special software is used to find data on the disc, by using the index, which is also stored on the disc, with the data itself.

The publisher's data is transfered to 9 track, half-inch computer tape, the standard means of transporting large quantities of data in the mainframe business, and shipped to the CD plant.

There, the data is read from the tapes onto large hard disks. The error correction codes and header information is added. The mainframe is connected to a laser beam recorder, where the data is used to modulate a laser beam, to create the series of pits in the master disc surface.

The master disc is then coated with aluminum so it can reflect the laser beam. It is checked on an inspection player. After the master is verified as free of mastering errors, it is used to produce stampers. The stampers are negative images of the CD ROM.

The stampers are used to mold the CD ROMs from polycarbonate, a strong plastic. Most CD plants use a form of injection molding. The molded CDs are then coated with aluminum and sealed with a protective coating. Then the label information is printed on the disc. All of the above steps are carried out in a class 100 clean room, the same level of air quality needed in integrated circuit manufacturing.

•AC•

Amiga BBS Numbers:

by Richard Rae
CIS: 72177,3516

DISCLAIMER: This is merely a compilation of information appearing on CIS and from other sources. No representations as to accuracy are expressed or implied.

Location	Name	Phone Number	Hours	Baud	Notes
AZ Phoenix	Amiga Talk	602-846-3901	24 HOUR	300	
CA Los Angeles	AmigaBoard	213-478-9788		300/1200	Beta testing BBS Software
CA	Developers Exchange	408-372-1722		1200	MaxiCorp Board
CA	THE WELL	415-332-6106		1200	Programmer Network
CA San Carlos	FAUG	415-595-5452	6PM-7AM	300	24 HOUR on weekends
CA	RSVP	415-659-9169		1200	Fidonet node
CA San Francisco	WELCHNET	415-664-2811		1200	Fidonet node
CA	BBS-JC	415-961-7250	24 HOUR	300/1200/2400	Amiga section
CA Del Mar	Dreamscape	619-755-2863	24 HOUR	300/1200/2400	
CA San Clemente	VIVID XPRESS	714-493-6094	24 HOUR	300/1200	BBS source available
CA Costa Mesa	Amiga's Den	714-646-2723	24 HOUR		
CA Irvine		714-660-1462			
CA Amiga Xpress		805-397-4812	24 HOUR	300/1200	
CA Ventura	VTBBS	805-656-3746	24 HOUR	300/1200/2400	Amiga section
CA Santa Barbra	Compucation	805-967-0895			
CO CO Springs	SHAMUS ST	303-597-7543	24 HOUR	300/1200	Amiga/Atari ST board
CO Denver	A.B.C.	303-693-4735	24 HOUR	300/1200/2400	
DE	Beagle Nest	302-731-7842	10pm-10am	300/1200/2400	Amiga is section 6
DE Wilmington	PC-NUG Board	302-737-2294	24 HOUR	300/1200	Amiga is section 21
FL Miami	MIAMI AMIGA	305-255-5307	24 HOUR	300/1200	
FL Ft. Lauderdale	AMIGAMAIL	305-486-7021	9PM-8AM	MSS BBS-PC on Amiga	
FL		305-737-1590		1200	
FL Jacksnville	CASA MI AMY	904-733-4515	24 HOUR	1200	Fidonet node
IL		312-357-2089			
IL GlenElyn	Lattice	312-858-8087		300/1200/2400	Fidonet node
IL	Wovsed 200	618-378-2133	5PM-8AM	300/1200	Use "Username:GUEST"
KS	MEGASTAR	316-251-7417	5PM-6AM	300/1200	
MD Baltimore		301-254-8078	24 HOUR	1200	
MI	Novi Downld	313-348-4479	24 HOUR	300/1200	348-4477 (Voice) 1st!
MO	St Louis Systems+	314-361-1270	24 HOUR		
MO		816-474-1052		1200	Amiga is section 14
NB		506-357-9660			
NC	MMS	919-779-6674	24 HOUR	300/1200/2400	Amiga section
NC	CompTel Cntr	919-832-7282	24 HOUR	300/1200	
NE Omaha	Wind Dragon	402-291-8053	24 HOUR	300/1200	Fidonet 14/614
NH Laconia	Sanctuary	603-524-0136	24 HOUR	300/1200	
NJ Englishton	AMY PALACE	201-446-1424	24 HOUR	300/1200	Amiga, Atari and Mac
NY New York	AMUSE	212-269-4879		300/1200/2400	Fidonet 107/34
NY		212-534-2858	1AM-6PM		
NY New York	68000 BBS	212-927-6919			68000 General Board
NY Bartwater	AMY ADVANTAGE	516-661-4881		300/1200	Amiga Advantage Board
NY	5th Generation	716-377-3985	24 HOUR	300/1200	
NY	MUSICNET	914-442-4006		300/1200	MIDI Board, $75/YR
OH Dayton	AmigaBoard	513-898-0702	24 HOUR	300/1200	
OH Columbus	Earthrise	614-868-1100			
TN	Connection	615-868-7860	24 HOUR	300/1200	
TX Dallas	Rising Star	214-231-1372		300/1200/2400	Fidonet 124/15.Sect.3
TX Dallas	Nomads Nook	817-926-8922		300	
UT SaltLake	TechniSoft	801-264-8290	24 HOUR	1200	Fidonet 15/464
WA		206-874-6381	24 HOUR	300/1200	Fidonet node
WY	Black Diamond	307-682-7987	24 HOUR	300/1200	

Amazing Computing™

Index of Advertisers

COMING NEXT MONTH!

in

Amazing Computing

June

News From COMDEX Spring: all the goodies

Amazing tutorials:

The Amazing C Tutorial
Forth
The Amazing AmigaBasic™ Tutorial

ROOMERS
Miga-Mania.
The AMICUS™ Network
New software and hardware releases for the Amiga™.
New Amiga™ public domain software
and, as always, a few surprises!

BECAUSE YOU'RE GOING TO BUY ONLY ONE DISK BACKUP UTILITY FOR YOUR

AMIGA

MAKE IT MARAUDER®

MARAUDER IS EVERYTHING YOU EXPECT

Marauder works on any standard Amiga with 256K of memory and one disk drive.
Marauder backs up most copy-protected software titles.
Marauder's user interface is clear-cut, easy to understand, and easy to use.
Marauder's manual is written in plain English without confusing technical terms.
Marauder insures that your valuable software investment will be protected against theft, loss and accidental erasure.

MARAUDER IS EVERYTHING YOU NEED

Marauder takes advantage of Amiga's optional external disk drive and extra RAM if you have them.
Maurauder, itself, is not copy protected in any way.
Marauder uses Amiga's advanced graphics and processing power to its fullest advantage.
Marauder automatically formats your backup disk while copying.
Many complicated protection schemes are easily handled by Marauder's parameter-entry mode — two numbers is all it takes.
Current parameters for all titles will be available **FREE** to registered users at any time.

BEST OF ALL MARAUDER is AVAILABLE NOW!!!

You can get Marauder at the
SPECIAL INTRODUCTORY PRICE OF

MasterCard **ONLY $39.95** VISA

call now (215) 546-1533

(Dealer Inquiries Welcome)

DISCOVERY SOFTWARE

262 South 15th Street • Suite 300 • Philadelphia, PA 19102-3801

WriteHand

BYTE BY BYTE INTRODUCES WRITE HAND FOR THE AMIGA

WRITE HAND is a general word processor and form letter generator that gives you the most features for your dollars. Developed to meet the special needs of small business, WRITE HAND is easy to learn and easy to use.

WRITE HAND challenges you to compare the following features dollar-for-dollar, feature-for-feature to those of other word processors on the market today.

- Extensive on-line HELP service
- Form letter generator
- Powerful editing capabilities
- Formats documents while you edit
- Reviews and merges files while you edit
- Moves blocks of text and figures of any size
- Provides word wrap, bolding and underlining

Make WRITE HAND the tool that moves your business into the productive world of electronic word processing.

Suggested Retail Price: $50.00 Call or mail orders to:

BYTE by BYTE™ CORPORATION

3736 Bee Cave Rd., Suite 3
Austin, TX 78746

(512) 328-2985

Major credit cards and CODs accepted. Certified checks and money orders *only*.

FinancialPlus

BYTE BY BYTE INTRODUCES FINANCIAL PLUS FOR THE AMIGA.

FINANCIAL PLUS is the affordable way to put your business at your fingertips. FINANCIAL PLUS is the complete accounting solution with five systems in one:

- General Ledger
- Accounts Payable
- Accounts Receivable
- Payroll
- Word Processor

FINANCIAL PLUS is adaptable. You customize each company according to its size and bookkeeping needs.

An easy-to-read, easy-to-learn users guide provides comprehensive instructions for setting up your own books. Plain-English menus are the system "roadmaps" for both the novice and for the more experienced.

Because FINANCIAL PLUS is a totally integrated accounting system, no longer must you purchase individual packages, store entries on separate diskettes, or run confusing transfer programs to obtain complete integration.

Suggested Retail Price: $295.00 Call or mail orders to:

BYTE by BYTE™ CORPORATION

3736 Bee Cave Rd., Suite 3
Austin, TX 78746

(512) 328-2985

Major credit cards and CODs accepted. Certified checks and money orders *only*.